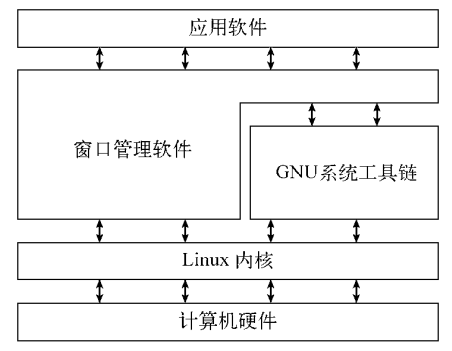
# 第1章：初识Linux Shell

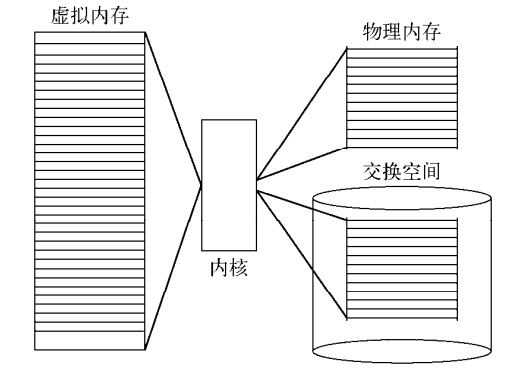
1. 【page 2】linux组成：



1. 【page 3】**内核主要负责**以下四种功能：

（1）系统内存管理：不仅管理服务器上的可用物理内存，还可以创建和管理虚拟内存   
（2）软件程序管理：内核控制着Linux系统如何管理运行在系统上的所有进程。   
（3）硬件设备管理：驱动代码包括编译进内核的设备驱动代码和可插入内核的设备驱动模块。  
（4）文件系统管理：内核必须在编译时就加入对所有可能用到的文件系统的支持。

3、【page 4】内核通过硬盘上的存储空间来实现虚拟内存，这块区域称为交换空间（ swap space）。



4、【page 5】内核创建了第一个进程（称为init进程）来启动系统上所有其他进程。当内核启动时，它会将init进程加载到虚拟内存中。 一些Linux发行版使用一个表来管理在系统开机时要自动启动的进程。在Linux系统上，这个表通常位于专门文件/etc/inittab中。

另外一些系统（比如现在流行的Ubuntu Linux发行版）则采用/etc/init.d目录，将开机时启动或停止某个应用的脚本放在这个目录下。这些脚本通过/etc/rcX.d目录下的入口（ entry） 启动，这里的X代表运行级（ run level）。

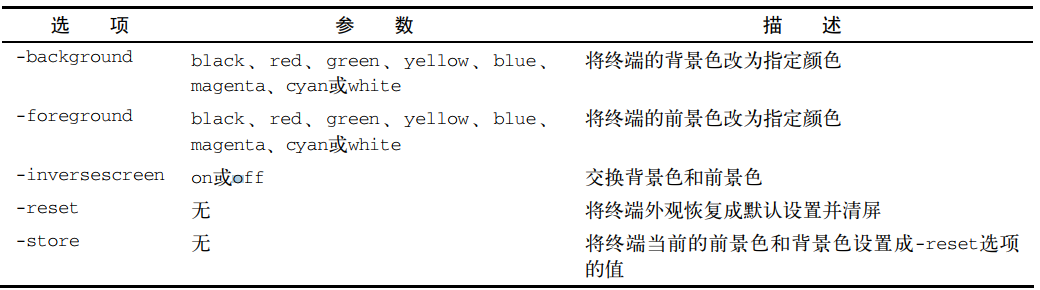
1. 【page 7】GUN系统工具链由shell和核心GNU工具组成，供Linux系统使用的这组核心工具被称为coreutils（core utilities）软件包。由三部分构成：

（1）用以处理文件的工具  
（2）用以操作文本的工具  
（3）用以管理进程的工具

GNU/Linux shell是一种特殊的交互式工具。shell的核心是命令行提示符。

# 第2章：走进shell

1. 【page 18】Linux发行版通常使用Ctrl+Alt组合键配合F1或F7来进入图形界面。 tty代表电传打字机（ teletypewriter）。这是一个古老的名词，指的是一台用于发送消息的机器。
2. 【page 20】setterm命令设置终端的文本颜色和背景，将终端的背景色设置成白色、文本设置成黑色，方法是输入命令setterm -inversescreen on，或者setterm –background white然后setterm –foreground black。



# 第3章：基本的bash shell命令

1、GNU bash shell能提供对Linux系统的交互式访问。它是作为普通程序运行的，通常是在用户登录终端时启动。登录时系统启动的shell依赖于用户账户的配置。 /etc/passwd文件包含了所有系统用户账户列表以及每个用户的基本配置信息。

2、【page35】man命令用来访问存储在Linux系统上的手册页面。 上下键切换浏览内容，空格键可以翻页。q键退出浏览。

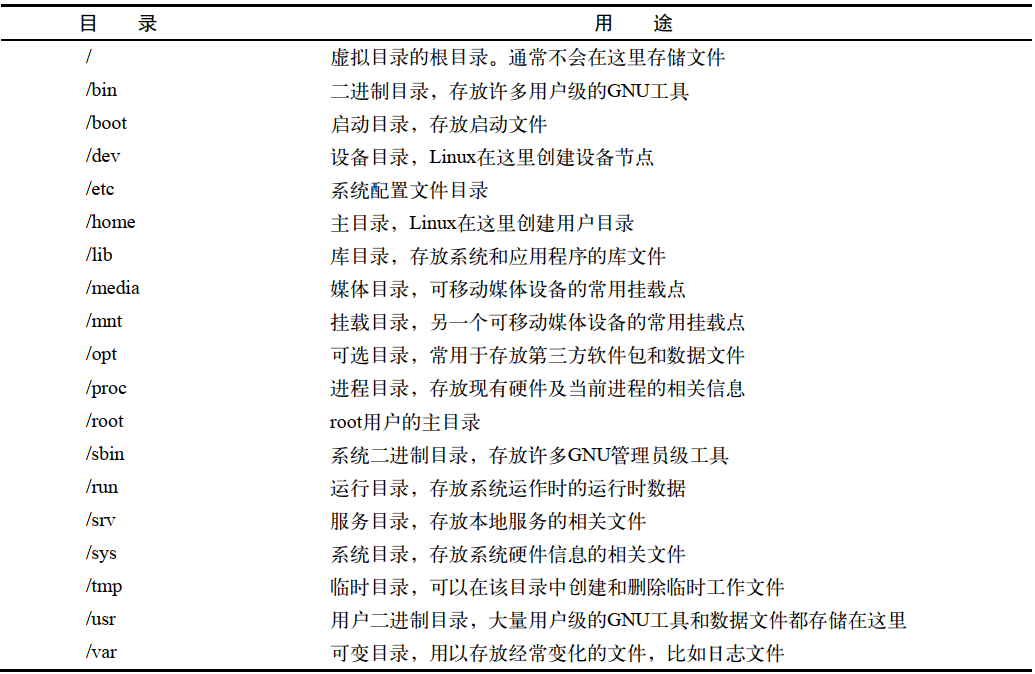
如果不记得命令名怎么办？可以使用关键字搜索手册页。语法是： man -k 关键字。例如，要查找与终端相关的命令，可以输入man -k terminal。 man工具通常提供的是命令所对应的最低编号的内容，每个相关命令的区域号代表：

1 可执行程序或shell命令  
2 系统调用  
3 库调用  
4 特殊文件  
5 文件格式与约定

6 游戏  
7 概览、约定及杂项  
8 超级用户和系统管理员命令  
9 内核例程

3、你将会发现Linux使用正斜线（ /）而不是反斜线（ \）在文件路径中划分目录。在Linux中，反斜线用来标识转义字符，要是用在文件路径中的话会导致各种各样的问题。

4、【page 39】Linux会在根驱动器上创建一些特别的目录，我们称之为挂载点（ mount point）。挂载点是虚拟目录中用于分配额外存储设备的目录。虚拟目录会让文件和目录出现在这些挂载点目录中，然而实际上它们却存储在另外一个驱动器中。



5、pwd命令可以显示出shell会话的当前目录，这个目录被称为当前工作目录。在切换到新的当前工作目录时使用pwd命令，是很好的习惯。

相对文件路径允许用户指定一个基于当前位置的目标文件路径。相对文件路径不以代表根目录的正斜线（ /）开头，而是以目录名（如果用户准备切换到当前工作目录下的一个目录）或是一个特殊字符开始。

1. 【page40】cd命令遍历目录：

cd #切换到主目录

cd . #当前目录

cd .. #当前目录的上级目录

cd ~ #切换主目录

7、【page 43】ls命令显示当前目录下的文件和目录：

-R #递归列出所有

-l #参数会产生长列表格式的输出，包含了目录中每个文件的更多相关信息。

8、文件处理命令：

（1）touch命令创建了你指定的新文件，并将你的用户名作为文件的属主。

（2）【page 47】cp命令复制文件：cp source destination

-i选项，强制shell询问是否需要覆盖已有文件

-R参数威力强大。可以用它在一条命令中递归地复制整个目录的内容。

（3）mv命令可以将文件和目录移动到另一个位置或重新命名

mv filename newname - inode编号和时间戳保持不变。这是因为mv只影响文件名。

mv path newpath #剪切

mv path/filename path/newfilename #剪切并重命名

（4）rm命令删除文件：rm file

-i命令参数提示你是不是要真的删除该文件。

-f参数强制删除

（5）要为一个文件创建符号链接，原始文件必须事先存在。然后可以使用ln命令以及-s选项来创建符号链接。

ln -s file link\_file

如果使用cp命令复制一个文件，而该文件又已经被链接到了另一个源文件上，那么你得到的其实是源文件的一个副本(不是链接文件的副本)。

9、目录处理命令

（1）mkdir命令创建目录：mkdir New\_Dir

要想同时创建多个目录和子目录，需要加入-p参数

（2）rmdir命令删除目录。

默认情况下， rmdir命令只删除空目录。-r选项使得命令可以向下进入目录，删除其中  
的文件，然后再删除目录本身但是有时牵扯到权限问题会询问。

一口气删除目录及其所有内容的终极大法就是使用带有-rf参数的rm命令。

1. 查看文件

（1）file命令是一个随手可得的便捷工具。 它能够探测文件的内部，并决定文件是什么类型的：fime filename

（2）cat命令是显示文本文件中所有数据的得力工具。

-n参数会给所有的行加上行号，如果只想给有文本的行加上行号，可以用-b参数。

如果不想让制表符出现，可以用-T参数。

1. cat命令的主要缺陷是：一旦运行，你就无法控制后面的操作。more命令会显示文本文件的内容，但会在显示每页数据之后停下来。
2. less命令的操作和more命令基本一样，一次显示一屏的文件文本。除了支持和more命令相同的命令集，它还包括更多的选项。

（5）tail命令会显示文件最后几行的内容（文件的“尾部”）。默认情况下，它会显示文件的末尾10行。 可以向tail命令中加入-n参数来修改所显示的行数：tail -n x file

-f参数是tail命令的一个突出特性。它允许你在其他进程使用该文件时查看文件的内容。  
tail命令会保持活动状态，并不断显示添加到文件中的内容。这是实时监测系统日志的绝妙  
方式。

（6）head命令会显示文件开头那些行的内容。默认情况下，它会显示文件前10行的文本。用法同tail，也可实时监测系统日志。

# 第4章：更多的bash shell

1、【page64】ps命令探查进程

（1）Unix风格的参数，前面加单破折线（常用）:

-A 显示所有进程

-a 显示除控制进程（session leader①）和无终端进程外的所有进程

-e 显示所有运行在系统上的进程

-G grplist 显示组ID在grplist列表中的进程

-U userlist 显示属主的用户ID在userlist列表中的进程

-p pidlist 显示PID在pidlist列表中的进程

-t ttylist 显示终端ID在ttylist列表中的进程

-u userlist 显示有效用户ID在userlist列表中的进程

-F 显示更多额外输出（相对-f参数而言）

-O format 显示默认的输出列以及format列表指定的特定列

-M 显示进程的安全信息

-c 显示进程的额外调度器信息

-f 显示完整格式的输出

-j 显示任务信息

-l 显示长列表

-H 用层级格式来显示进程（树状，用来显示父进程）

-L 显示进程中的线程

-V 显示ps命令的版本号

ps -f详细信息：

 UID：启动这些进程的用户。

 PID：进程的进程ID。

 PPID：父进程的进程号（如果该进程是由另一个进程启动的）。

 C：进程生命周期中的CPU利用率。

 STIME：进程启动时的系统时间。

 TTY：进程启动时的终端设备。

 TIME：运行进程需要的累计CPU时间。

 CMD：启动的程序名称。

ps -l详细信息：

 F：内核分配给进程的系统标记。

 S：进程的状态（O代表正在运行；S代表在休眠；R代表可运行，正等待运行；Z代表僵化，进程已结束但父进程已不存在；T代表停止）。

 PRI：进程的优先级（越大的数字代表越低的优先级）。

 NI：谦让度值用来参与决定优先级。

 ADDR：进程的内存地址。

 SZ：假如进程被换出，所需交换空间的大致大小。

 WCHAN

（2）GNU长参数:

2、【page 71】top命令实时监测进程

（1）第一行显示了当前时间、系统的运行时间、登录的用户数以及系统的平均负载。平均负载有3个值：最近1分钟的、最近5分钟的和最近15分钟的平均负载。

（2）第二行显示了进程概要信息——top命令的输出中将进程叫作任务（task）：有多少进程处在运行、休眠、停止或是僵化状态（僵化状态是指进程完成了，但父进程没有响应）。

（3）第三行显示了CPU的概要信息。top根据进程的属主（用户还是系统）和进程的状态（运行、空闲还是等待）将CPU利用率分成几类输出。

（4）第4-5行说明了系统内存的状态。第一行说的是系统的物理内存：总共有多少内存， 当前用了多少，还有多少空闲。后一行说的是同样的信息，不过是针对系统交换空间（如果分配了的话）的状态而言的。

（5）最后一部分显示了当前运行中的进程的详细列表：

 PID：进程的ID。

 USER：进程属主的名字。

 PR：进程的优先级。

 NI：进程的谦让度值。

 VIRT：进程占用的虚拟内存总量。

 RES：进程占用的物理内存总量。

 SHR：进程和其他进程共享的内存总量。

 S：进程的状态（D代表可中断的休眠状态，R代表在运行状态，S代表休眠状态，T代表跟踪状态或停止状态，Z代表僵化状态）。

 %CPU：进程使用的CPU时间比例。

 %MEM：进程使用的内存占可用内存的比例。

 TIME+：自进程启动到目前为止的CPU时间总量。

 COMMAND：进程所对应的命令行名称，也就是启动的程序名。

1. 信号



4、kill命令可通过进程ID（PID）给进程发信号。默认情况下，kill命令会向命令行中列出的全部PID发送一个TERM信号。

TERM信号告诉进程可能的话就停止运行。不过，如果有不服管教的进程，那它通常会忽略这个请求。如果要强制终止，-s参数支持指定其他信号（用信号名或信号值）。

5、killall命令非常强大，它支持通过进程名而不是PID来结束进程。killall命令也支持通配符，这在系统因负载过大而变得很慢时很有用。

6、Linux文件系统将所有的磁盘都并入一个虚拟目录下。在使用新的存储媒体之前，需要把它放到虚拟目录下。这项工作称为挂载（mounting）。

7、【page 73】mount命令用来输出当前系统上挂载的设备列表，挂载媒体。

（1）提供如下四部分信息：

 媒体的设备文件名

 媒体挂载到虚拟目录的挂载点

 文件系统类型

 已挂载媒体的访问状态

（2）要手动在虚拟目录中挂载设备，需要以root用户身份登录，或是以root用户身份运行sudo命令：mount -t type device directory，type参数指定了磁盘被格式化的文件系统类型。

（3）其他常用选项

-a 挂载/etc/fstab文件中指定的所有文件系统

-f 使mount命令模拟挂载设备，但并不真的挂载

-F 和-a参数一起使用时，会同时挂载所有文件系统

-v 详细模式，将会说明挂载设备的每一步

-I 不启用任何/sbin/mount.filesystem下的文件系统帮助文件

-l 给ext2、ext3或XFS文件系统自动添加文件系统标签

-n 挂载设备，但不注册到/etc/mtab已挂载设备文件中

-p num 进行加密挂载时，从文件描述符num中获得密码短语

-s 忽略该文件系统不支持的挂载选项

-r 将设备挂载为只读的

-w 将设备挂载为可读写的（默认参数）

-L label 将设备按指定的label挂载

-U uuid 将设备按指定的uuid挂载

-O 和-a参数一起使用，限制命令只作用到特定的一组文件系统上

-o 给文件系统添加特定的选项

-o参数允许在挂载文件系统时添加一些以逗号分隔的额外选项。以下为常用的选项：

 ro：以只读形式挂载。

 rw：以读写形式挂载。

 user：允许普通用户挂载文件系统。

 check=none：挂载文件系统时不进行完整性校验。

 loop：挂载一个文件。

8、umount命令卸载设备，从Linux系统上移除一个可移动设备时，不能直接从系统上移除，而应该先卸载，如果有任何程序正在使用设备上的文件，系统就不会允许你卸载它。

umount [directory | device ]

9、df命令可以让你很方便地查看所有已挂载磁盘的使用情况。

 设备的设备文件位置；

 能容纳多少个1024字节大小的块；

 已用了多少个1024字节大小的块；

 还有多少个1024字节大小的块可用；

 已用空间所占的比例；

 设备挂载到了哪个挂载点上。

一个常用的参数是-h。它会把输出中的磁盘空间按照用户易读的形式显示，通常用M来替代兆字节，用G替代吉字节。

10、du命令可以显示某个特定目录（默认情况下是当前目录）的磁盘使用情况。这一方法可用来快速判断系统上某个目录下是不是有超大文件。

du [directory]

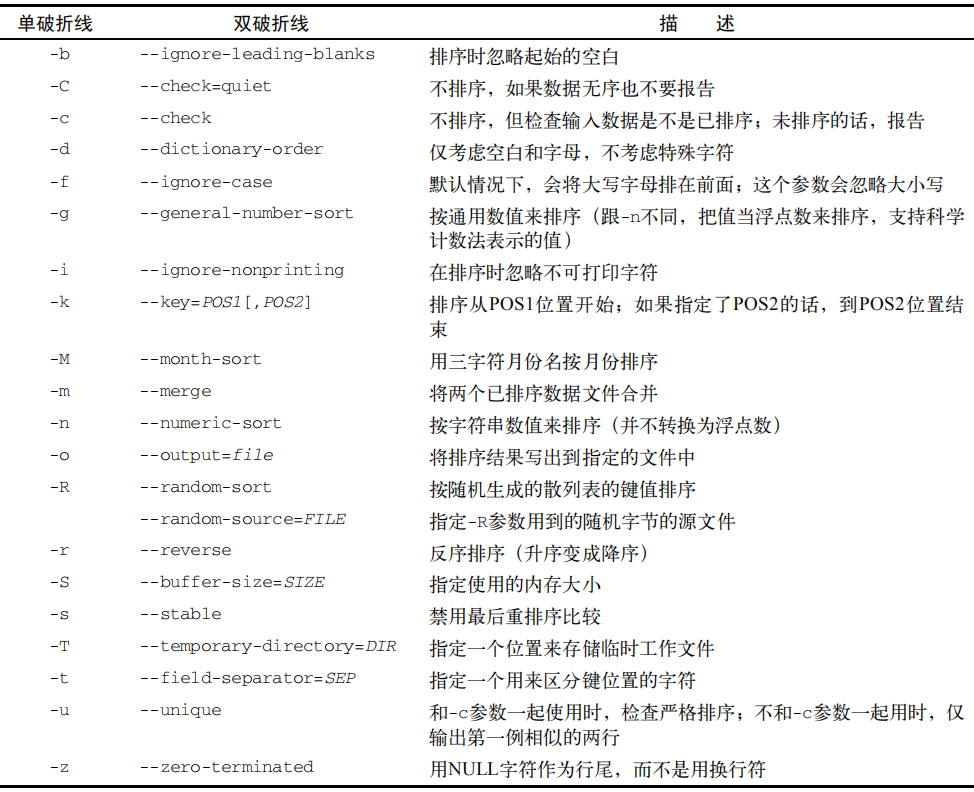
 -c：显示所有已列出文件总的大小。

 -h：按用户易读的格式输出大小，即用K替代千字节，用M替代兆字节，用G替代吉字

节。

 -s：显示每个输出参数的总计。

11、【page 78】sort命令按照会话指定的默认语言的排序规则对文本文件中的数据行排序.



-n参数告诉sort命令把数字识别成数字而不是字符，并且按值排序。默认情况下，sort命令会把数字当做字符来执行标准的字符排序。

可以用-t参数来指定字段分隔符，然后用-k参数来指定排序的字段。

12、【page 81】grep命令搜索数据，grep命令会在输入或指定的文件中查找包含匹配指定模式的字符的行。grep的输出就是包 含了匹配模式的行。

grep [options] pattern [file]

grep [options] [正则] [file]

如果要进行反向搜索（输出不匹配该模式的行），可加-v参数；

如果要显示匹配模式的行所在的行号，可加-n参数；

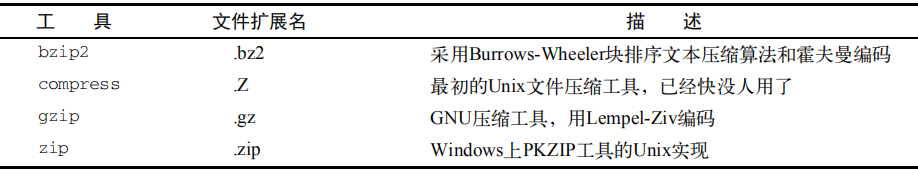
如果只要知道有多少行含有匹配的模式，可用-c参数；

如果要指定多个匹配模式，可用-e参数来指定每个模式；

egrep命令是grep的一个衍生，支持POSIX扩展正则表达式。

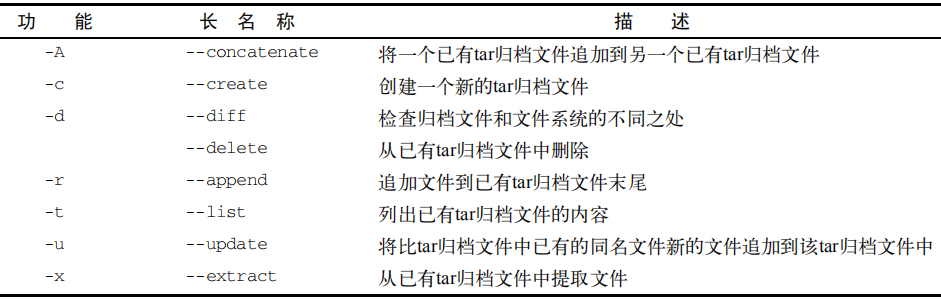
fgrep命令是另外一个版本，支持将匹配模式指定为用换行符分隔的一列固定长度的字符串。

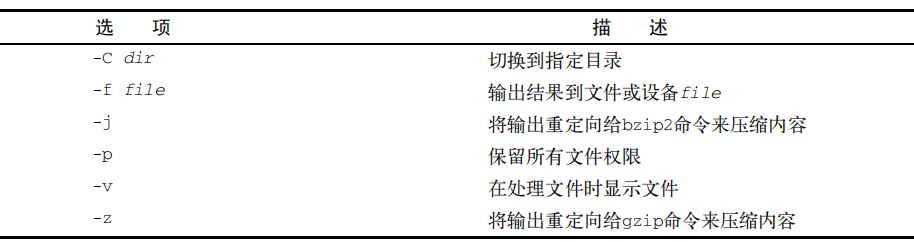
13、Linux文件压缩工具



14、【page 84】tar命令最开始是用来将文件写到磁带设备上归档的，然而它也能把输出写到文件里，这种用法在Linux上已经普遍用来归档数据了。

tar function [options] object1 object2 ...





常用组合：

压缩：tar -cvf test.tar test/ test2/

解压：tar -xvf test.tar

# 第5章：理解shell

1、在/etc/passwd文件中，在用户ID记录的第7个字段中列出了默认的shell程序。只要用户登录到某个虚拟控制台终端或是在GUI中启动终端仿真器，默认的shell程序就会开始运行。

用户默认的交互shell是/bin/bash，另外一个默认shell是/bin/sh，它作为默认的系统shell，用于那些需要在启动时使用的系统shell脚本。

2、用于登录某个虚拟控制器终端或在GUI中运行终端仿真器时所启动的默认的交互shell，是一个父shell。在CLI提示符后输入/bin/bash命令或其他等效的bash命令时，会创建一个新的shell程序，这个shell程序被称为子shell。在生成子shell进程时，只有部分父进程的环境被复制到子shell环境中。

在生成子shell的前后ps配合选项-f来使用查看到或者使用ps --forest，使用exit可以退出子shell。在父shell中输入exit，能够退出CLI。

3、命令列表要想成为进程列表，这些命令必须包含在括号里，生成了一个子shell来执行对应的命令，例如：

$ **(pwd ; ls ; cd /etc ; pwd ; cd ; pwd ; ls)**

进程列表是一种命令分组（command grouping）。另一种命令分组是将命令放入花括号中，并在命令列表尾部加上分号（;）。语法为{ command; }。使用花括号进行命令分组并不会像进程列表那样创建出子shell。

要想知道是否生成了子shell，得借助命令echo $BASH\_SUBSHELL。如果该命令返回0，就表明没有子shell。如果返回1或者其他更大的数字，就表明存在子shell。

例如：( pwd ; (echo $BASH\_SUBSHELL)) 会输出当前目录和2；

进程列表、协程和管道都利用了子shell。

4、一个高效的子shell用法就是使用后台模式。要想将命令置入后台模式，可以在命令末尾加上字符&。

jobs命令可以显示出当前运行在后台模式中的所有用户的进程（作业）。-l选项还能够看到更多的相关信息。

使用tar创建备份文件是有效利用后台进程列表的一个更实用的例子：

(tar -cf xx.tar dir1/ dir2/)&

6、协程就是将进程列表置入后台模式的另一种方法。协程可以同时做两件事。它在后台生成一个子shell，并在这个子shell中执行命令。使用coproc命令创建协程。

coproc cmd 或者 coproc My\_Job { cmd; } My\_Job是协程名字。

必须确保在第一个花括号（{）和命令名之间有一个空格。还必须保证命令以分号（;）结

尾。另外，分号和闭花括号（}）之间也得有一个空格。

7、外部命令，有时候也被称为文件系统命令，是存在于bash shell之外的程序。它们并不是shell程序的一部分。外部命令程序通常位于/bin、/usr/bin、/sbin或/usr/sbin中。例如ps。

which和type命令可以找到命令的所在位置。

当外部命令执行时，会创建出一个子进程。这种操作被称为衍生（forking）。外

8、内建命令和外部命令的区别在于前者不需要使用子进程来执行。它们已经和shell编译成了一体，作为shell工具的组成部分存在。不需要借助外部程序文件来运行。例如cd。

内建命令的执行速度要更快，效率也更高。

有些命令有多种实现。例如echo和pwd既有内建命令也有外部命令。命令type -a显示出了每个命令的两种实现。

9、一个有用的内建命令是history命令。bash shell会跟踪你用过的命令。你可以唤回这些命令并重新使用。你可以设置保存在bash历史记录中的命令数。要想实现这一点，你需要修改名为HISTSIZE的环境变量。

输入!!，然后按回车键就能够唤出刚刚用过的那条命令来使用，!N使用第N条。

命令历史记录被保存在隐藏文件.bash\_history中，它位于用户的主目录中。bash命令的历史记录是先存放在内存中，当shell退出时才被写入到历史文件中。

可以在退出shell会话之前强制将命令历史记录写入.bash\_history文件。要实现强制写入，需

要使用history -a。如果你打开了多个终端会话，仍然可以使用history -a命令在打开的会话中 向.bash\_history文件中添加记录。但是对于其他打开的终端会话，历史记录并不会自动更新。这是因为.bash\_history文件只有在打开首个终端会话时才会被读取。要想强制重新读

取.bash\_history文件，更新终端会话的历史记录，可以使用history -n命令。

10、alias命令是另一个shell的内建命令。命令别名允许你为常用的命令（及其参数）创建另一个名称，从而将输入量减少到最低。

查看当前可用的别名，使用alias -p;

可以使用alias命令创建属于自己的别名，例如: $ alias li='ls -li'

# 第6章：使用Linux环境变量

1、bash shell用一个叫作环境变量（environment variable）的特性来存储有关shell会话和工作环境的信息（这也是它们被称作环境变量的原因）。这项特性允许你在内存中存储数据，以便程序或shell中运行的脚本能够轻松访问到它们。

全局环境变量对于shell会话和所有生成的子shell都是可见的。局部变量则只对创建它们的shell可见。

要查看全局变量，可以使用env或printenv或export命令。

set命令会显示为某个特定进程设置的所有环境变量，包括局部变量、全局变量以及用户定义变量。

2、设置局部用户定义变量

例如：my\_variable=’Hello world’

如果要给变量赋一个含有空格的字符串值，必须用单引号来界定字符串的首和尾。

所有的环境变量名均使用大写字母，这是bash shell的标准惯例。如果是你自己创建的局部变量或是shell脚本，请使用小写字母。

如果你在子进程中设置了一个局部变量，那么一旦你退出了子进程，那个局部环境变量就不可用。当我们回到父shell时，子shell中设置的局部变量就不存在了。

3、过export命令设置全局环境变量

例如：my\_variable=’Hello world’;export my\_variable;

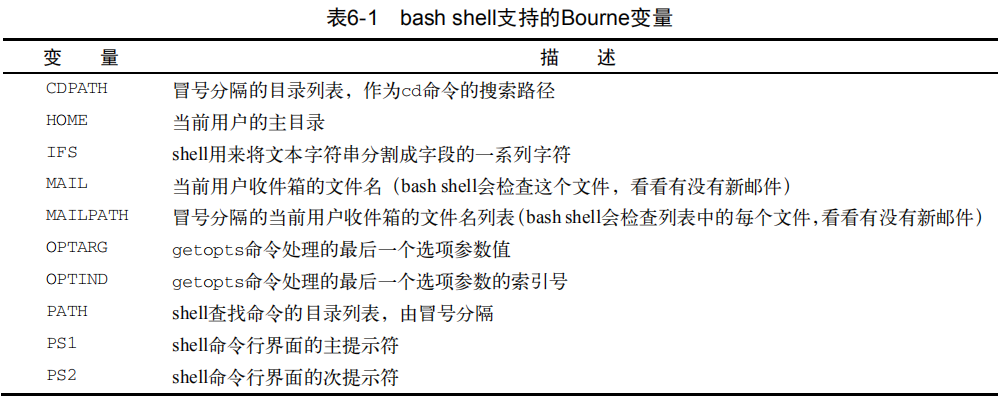
或者 export my\_variable=’Hello world’

修改子shell中全局环境变量并不会影响到父shell中该变量的值。子shell甚至无法使用export命令改变父shell中全局环境变量的值。

4、用unset命令删除已经存在的环境变量。

如果要用到变量，使用$；如果要操作变量，不使用$。

5、【page 110】默认的 shell 环境变量



bash shell还提供一些自有的变量：

BASH 当前shell实例的全路径名

BASH\_ALIASES 含有当前已设置别名的关联数组

BASH\_ARGC 含有传入子函数或shell脚本的参数总数的数组变量

BASH\_ARCV 含有传入子函数或shell脚本的参数的数组变量

BASH\_CMDS 关联数组，包含shell执行过的命令的所在位置

BASH\_COMMAND shell正在执行的命令或马上就执行的命令

BASH\_ENV 设置了的话，每个bash脚本会在运行前先尝试运行该变量定义的启动文件

BASH\_EXECUTION\_STRING 使用bash -c选项传递过来的命令

BASH\_LINENO 含有当前执行的shell函数的源代码行号的数组变量

BASH\_REMATCH 只读数组，在使用正则表达式的比较运算符=~进行肯定匹配（positive match）时，包 含了匹配到的模式和子模式

BASH\_SOURCE 含有当前正在执行的shell函数所在源文件名的数组变量

BASH\_SUBSHELL 当前子shell环境的嵌套级别（初始值是0）

BASH\_VERSINFO 含有当前运行的bash shell的主版本号和次版本号的数组变量

BASH\_VERSION 当前运行的bash shell的版本号

BASH\_XTRACEFD 若设置成了有效的文件描述符（0、1、2），则'set -x'调试选项生成的跟踪输出可被重 定向。通常用来将跟踪输出到一个文件中

BASHOPTS 当前启用的bash shell选项的列表

BASHPID 当前bash进程的PID

COLUMNS 当前bash shell实例所用终端的宽度

COMP\_CWORD COMP\_WORDS变量的索引值，后者含有当前光标的位置

COMP\_LINE 当前命令行

COMP\_POINT 当前光标位置相对于当前命令起始的索引

COMP\_KEY 用来调用shell函数补全功能的最后一个键

COMP\_TYPE 一个整数值，表示所尝试的补全类型，用以完成shell函数补全

COMP\_WORDBREAKS Readline库中用于单词补全的词分隔字符

COMP\_WORDS 含有当前命令行所有单词的数组变量

COMPREPLY 含有由shell函数生成的可能填充代码的数组变量

COPROC 占用未命名的协进程的I/O文件描述符的数组变量

DIRSTACK 含有目录栈当前内容的数组变量

EMACS 设置为't'时，表明emacs shell缓冲区正在工作，而行编辑功能被禁止

ENV 如果设置了该环境变量，在bash shell脚本运行之前会先执行已定义的启动文件（仅用 于当bash shell以POSIX模式被调用时）

EUID 当前用户的有效用户ID（数字形式）

FCEDIT 供fc命令使用的默认编辑器

FIGNORE 在进行文件名补全时可以忽略后缀名列表，由冒号分隔

FUNCNAME 当前执行的shell函数的名称

FUNCNEST 当设置成非零值时，表示所允许的最大函数嵌套级数（一旦超出，当前命令即被终止）

GLOBIGNORE 冒号分隔的模式列表，定义了在进行文件名扩展时可以忽略的一组文件名

GROUPS 含有当前用户属组列表的数组变量

histchars 控制历史记录扩展，最多可有3个字符

HISTCMD 当前命令在历史记录中的编号

HISTCONTROL 控制哪些命令留在历史记录列表中

HISTFILE 保存shell历史记录列表的文件名（默认是.bash\_history）

HISTFILESIZE 最多在历史文件中存多少行

HISTTIMEFORMAT 如果设置了且非空，就用作格式化字符串，以显示bash历史中每条命令的时间戳

HISTIGNORE 由冒号分隔的模式列表，用来决定历史文件中哪些命令会被忽略

HISTSIZE 最多在历史文件中存多少条命令

HOSTFILE shell在补全主机名时读取的文件名称

HOSTNAME 当前主机的名称

HOSTTYPE 当前运行bash shell的机器

IGNOREEOF shell在退出前必须收到连续的EOF字符的数量（如果这个值不存在，默认是1）

INPUTRC Readline初始化文件名（默认是.inputrc）

LANG shell的语言环境类别

LC\_ALL 定义了一个语言环境类别，能够覆盖LANG变量

LC\_COLLATE 设置对字符串排序时用的排序规则

LC\_CTYPE 决定如何解释出现在文件名扩展和模式匹配中的字符

LC\_MESSAGES 在解释前面带有$的双引号字符串时，该环境变量决定了所采用的语言环境设置

LC\_NUMERIC 决定着格式化数字时采用的语言环境设置

LINENO 当前执行的脚本的行号

LINES 定义了终端上可见的行数

MACHTYPE 用“CPU公司系统”（CPU-company-system）格式定义的系统类型

MAPFILE 一个数组变量，当mapfile命令未指定数组变量作为参数时，它存储了mapfile所读入 的文本

MAILCHECK shell查看新邮件的频率（以秒为单位，默认值是60）

OLDPWD shell之前的工作目录

OPTERR 设置为1时，bash shell会显示getopts命令产生的错误

OSTYPE 定义了shell所在的操作系统

PIPESTATUS 含有前台进程的退出状态列表的数组变量

POSIXLY\_CORRECT 设置了的话，bash会以POSIX模式启动

PPID bash shell父进程的PID

PROMPT\_COMMAND 设置了的话，在命令行主提示符显示之前会执行这条命令

PROMPT\_DIRTRIM 用来定义当启用了\w或\W提示符字符串转义时显示的尾部目录名的数量。被删除的目录 名会用一组英文句点替换

PS3 select命令的提示符

PS4 如果使用了bash的-x选项，在命令行之前显示的提示信息

PWD 当前工作目录

RANDOM 返回一个0～32767的随机数（对其的赋值可作为随机数生成器的种子）

READLINE\_LINE 当使用bind –x命令时，存储Readline缓冲区的内容

READLINE\_POINT 当使用bind –x命令时，表示Readline缓冲区内容插入点的当前位置

REPLY read命令的默认变量

SECONDS 自从shell启动到现在的秒数（对其赋值将会重置计数器）

SHELL bash shell的全路径名

SHELLOPTS 已启用bash shell选项列表，列表项之间以冒号分隔

SHLVL shell的层级；每次启动一个新bash shell，该值增加1

TIMEFORMAT 指定了shell的时间显示格式

TMOUT select和read命令在没输入的情况下等待多久（秒为单位）。默认值为0表示无限长

TMPDIR 目录名，保存bash shell创建的临时文件

UID 当前用户的真实用户ID（数字形式）

6、PATH环境变量定义了用于进行命令和程序查找的目录。

添加：PATH=$PATH:/home/christine/Scripts

7、在你登入Linux系统启动一个bash shell时，默认情况下bash会在几个文件中查找命令。这些文件叫作启动文件或环境文件。bash检查的启动文件取决于你启动bash shell的方式。启动bash shell有3种方式：

（1）登录时作为默认登录shell

 /etc/profile （主启动文件，Ubuntu中涉及/etc/bash.bashrc文件其包含了系统环境变量）

 $HOME/.bash\_profile（第二启动，会先去检查HOME目录中是不是还有一个叫.bashrc的启动文件。如果有的话，会先执行启动文件里面的命令。）

 $HOME/.bashrc

 $HOME/.bash\_login （第三启动）

 $HOME/.profile （第四启动）

（2）作为非登录shell的交互式shell

不是登录系统时启动的（比如是在命令行提示符下敲入bash时启动）。

 $HOME/.bashrc

.bashrc文件有两个作用：一是查看/etc目录下通用的bashrc文件，二是为用户提供一个定制自己的命令别名和私有脚本函数的地方。

（3）作为运行脚本的非交互shell

系统执行shell脚本时用的shell，它没有命令行提示符。

 BASH\_ENV环境变量。

当shell启动一个非交互式shell进程时，它会检查这个环境变量来查看要执行的启动文件。如果有指定的文件，shell会执行该文件里的命令，这通常包括shell脚本变量设置。

如果BASH\_ENV变量没有设置，子shell可以继承父shell导出过的变量。

8、永久修改环境变量

（1）在/etc/profile.d目录中创建一个以.sh结尾的文件。把所有新的或修改过的全局环境变量设置放在这个文件中。

（2）存储个人用户永久性bash shell变量的地方利用$HOME/.bashrc文件。如果设置了BASH\_ENV变量，那么记住，除非它指向的是$HOME/.bashrc，否则你应该将非交互式shell的用户变量放在别的地方。

9、数组变量：给某个环境变量设置多个值，可以把值放在括号里，值与值之间用空格分隔。

设置$ mytest=(one two three four five)

访问$ echo ${mytest[\*]}

删除$ unset mytest

# 第7章：理解Linux文件权限

1、Linux安全系统的核心是用户账户。用户权限是通过创建用户时分配的用户ID（User ID，通常缩写为UID）来跟踪的。

2、Linux系统使用一个专门的文件来将用户的登录名匹配到对应的UID值。这个文件就是/etc/passwd文件，它包含了一些与用户有关的信息。root用户账户是Linux系统的管理员，固定分配给它的UID是0。这些账户叫作系统账户，是系统上运行的各种服务进程访问资源用的特殊账户。Linux为系统账户预留了500以下的UID值。

/etc/passwd文件的7个字段包含了如下信息：

1.  登录用户名
2.  用户密码
3.  用户账户的UID（数字形式）
4.  用户账户的组ID（GID）（数字形式）
5.  用户账户的文本描述（称为备注字段）
6.  用户HOME目录的位置
7.  用户的默认

3、绝大多数Linux系统都将用户密码保存在另一个单独的文件中（叫作shadow文件，位置

在/etc/shadow）。只有特定的程序（比如登录程序）才能访问这个文件。

在/etc/shadow文件的每条记录中都有9个字段：

1.  与/etc/passwd文件中的登录名字段对应的登录名
2.  加密后的密码
3.  自上次修改密码后过去的天数密码（自1970年1月1日开始计算）
4.  多少天后才能更改密码
5.  多少天后必须更改密码
6.  密码过期前提前多少天提醒用户更改密码
7.  密码过期后多少天禁用用户账户
8.  用户账户被禁用的日期（用自1970年1月1日到当天的天数表示）
9.  预留字段给将来使用

4、【page 128】useradd命令使用系统的默认值以及命令行参数来设置用户账户。系统默认值被设置在/etc/default/useradd文件中。可以使用加入了-D选项的useradd命令查看所用Linux系统中的这些默认值。

GROUP=100 新用户会被添加到GID为100的公共组；

HOME=/home 新用户的HOME目录将会位于/home/loginname；

INACTIVE=-1 新用户账户密码在过期后不会被禁用；

EXPIRE= 新用户账户未被设置过期日期；

SHELL=/bin/bash 新用户账户将bash shell作为默认shell；

SKEL=/etc/skel 系统会将/etc/skel目录下的内容复制到用户的HOME目录下；

CREATE\_MAIL\_SPOOL=yes 系统为该用户账户在mail

/etc/skel目录下是bash shell环境的标准启动文件（.bashrc）。系统会自动将这些默认文件复制到你创建的每个用户的HOME目录。

useradd -m username //创建了新HOME目录，并将/etc/skel目录中的文件复制了过来

useradd -r username //创建系统账户

useradd -p passwd //为用户账户指定默认密码

useradd -u uid //为账户指定唯一的UID

5、userdel命令会只删除/etc/passwd文件中的用户信息，而不会删除系统中属于该账户的任何文件。如果加上-r参数，userdel会删除用户的HOME目录以及邮件目录。

6、usermod命令修改用户账户的字段，还可以指定主要组以及附加组的所属关系。

 -l修改用户账户的登录名。

 -L锁定账户，使用户无法登录。

 -p修改账户的密码。

 -U解除锁定，使用户能够登录。

7、用户管理与修改相关：

（1）passwd命令修改用户密码，只有root用户才有权限改别人的密码。

-e选项能强制用户下次登录时修改密码。

chpasswd命令可以为系统中的大量用户修改密码。能从标准输入自动读取登录名和密码对（由冒号分割）列表，给密码加密，然后为用户账户设置。你也可以用重定向命令来将含有userid:passwd对的文件重定向给该命令。

（2）chsh命令用来快速修改默认的用户登录shell。如：chsh -s /bin/csh test

（3）chfn命令提供了在/etc/passwd文件的备注字段中存储信息的标准方法。

（4）finger命令可以非常方便地查看Linux系统上的用户信息（该命令需要安装）。

（5）chage命令用来帮助管理用户账户的有效期。

-d 设置上次修改密码到现在的天数

-E 设置密码过期的日期

-I 设置密码过期到锁定账户的天数

-m 设置修改密码之间最少要多少天

-W 设置密码过期前多久开始出现提醒信息

（6）chage命令的日期值可以用YYYY-MM-DD格式的日期或者代表从1970年1月1日起到该日期天数的数值。

8、组权限允许多个用户对系统中的对象（比如文件、目录或设备等）共享一组共用的权限。/etc/group文件包含系统上用到的每个 组的信息。/etc/group文件有4个字段：

 组名

 组密码

 GID

 属于该组的用户列表

9、groupadd命令可在系统上创建新组，如果更改了已登录系统账户所属的用户组，该用户必须登出系统后再登录，组关系的更改才能生效。

groupmod命令可以修改已有组的GID（加-g选项）或组名（加-n选项）。修改组名时，GID和组成员不会变，只有组名改变。由于所有的安全权限都是基于GID的，你可以随意改变组名而不会影响文件的安全性。

10、【page 135】ls –l第一个字段就是描述文件和目录权限的编码。第一个字符：

 -代表文件

 d代表目录

 l代表链接

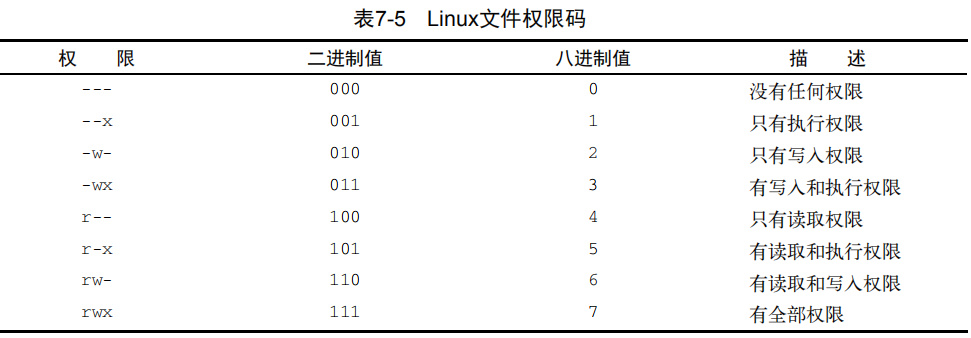
 c代表字符型设备

 b代表块设备

 n代表网络设备

其他字符依次代表：文件的属主、属组、其他用户的权限。

11、umask命令用来设置所创建文件和目录的默认权限。



12、chmod命令用来改变文件和目录的安全性设置。该命令的格式如下：

chmod options mode file

（1）mode参数使用八进制模式或符号模式进行安全性设置。

（2）mode参数使用符号模式的权限设置：

options = [ugoa…][[+-=][rwxXstugo…]

u代表用户，g代表组，o代表其他，a代表上述所有；

增加权限（+），移除权限（），或是将权限设置成后面的值（=）；

X：如果对象是目录或者它已有执行权限，赋予执行权限。

s：运行时重新设置UID或GID。

t：保留文件或目录。

ugo：将权限设置为跟属主/属组/其他用户一样。

13、chown命令用来改变文件的属主和属组。

chown options owner[.group] file

可用登录名或UID来指定文件的新属主。

-R选项配合通配符可以递归地改变子目录和文件的所属关系。

-h选项可以改变该文件的所有符号链接文件的所属关系。

除此之外，chgrp命令用来改变文件的默认属组。

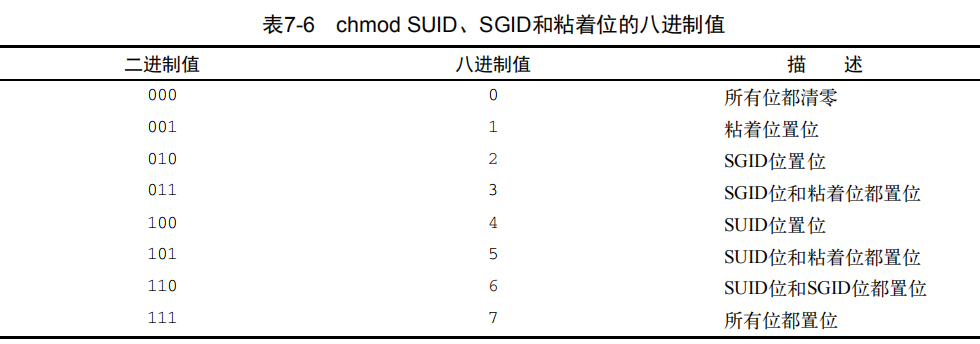
14、Linux还为每个文件和目录存储了3个额外的信息位。

 设置用户ID（SUID）：当文件被用户使用时，程序会以文件属主的权限运行。

 设置组ID（SGID）：对文件来说，程序会以文件属组的权限运行；对目录来说，目录中

创建的新文件会以目录的默认属组作为默认属组。

 粘着位：进程结束后文件还驻留（粘着）在内存中。



# 第8章：管理文件系统

1、ext文件系统

引入的最早的文件系统叫作扩展文件系统（extended filesystem，简记为ext）。它为Linux提供了一个基本的类Unix文件系统：使用虚拟目录来操作硬件设备，在物理设备上按定长的块来存储数据。ext文件系统采用名为索引节点的系统来存放虚拟目录中所存储文件的信息。索引节点系统在每个物理设备中创建一个单独的表（称为索引节点表）来存储这些文件的信息。

2、ext2文件系统

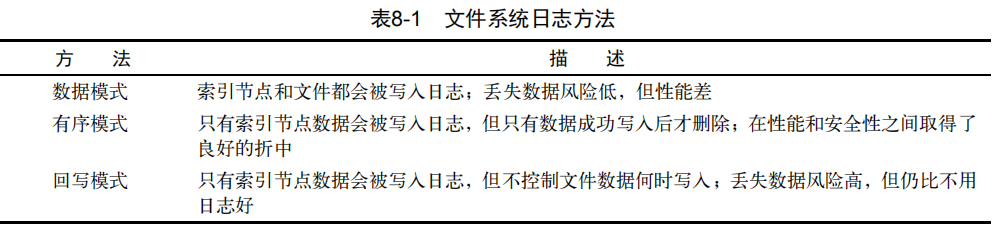
ext2文件系统是ext文件系统基本功能的一个扩展，但保持了同样的结构。ext2的索引节点表为文件添加了创建时间值、修改时间值和最后访问时间值来帮助系统管理 员追踪文件的访问情况。ext2文件系统还将允许的最大文件大小增加到了2 TB（在ext2的后期版本中增加到了32 TB），以容纳数据库服务器中常见的大文件。

ext2文件系统由于容易在系统崩溃或断电时损坏而臭名昭著。即使文件数据正常保存到了物理设备上，如果索引节点表记录没完成更新的话，ext2文件系统甚至都不知道那个文件存在！

3、日志文件系统为Linux系统增加了一层安全性。它不再使用之前先将数据直接写入存储设备再更新索引节点表的做法，而是先将文件的更改写入到临时文件（称作日志，journal）中。在数据成功写到存储设备和索引节点表之后，再删除对应的日志条目。

如果系统在数据被写入存储设备之前崩溃或断电了，日志文件系统下次会读取日志文件并处理上次留下的未写入的数据。

数据模式日志方法是目前为止最安全的数据保护方法，但同时也是最慢的。



4、ext3文件系统

它采用和ext2文件系统相同的索引节点表结构，但给每个存储设备增加了一个日志文件，以将准备写入存储设备的数据先记入日志。 默认情况下，ext3文件系统用有序模式的日志功能。

ext3文件系统无法恢复误删的文件，它没有任何内建的数据压缩功能（虽然有个需单独安装的补丁支持这个功能），ext3文件系统也不支持加密文件。

5、ext4文件系统

除了支持数据压缩和加密，ext4文件系统还支持一个称作区段（extent）的特性。区段在存储设备上按块分配空间，但在索引节点表中只保存起始块的位置。由于无需列出所有用来存储文件中数据的数据块，它可以在索引节点表中节省一些空间。

ext4还引入了块预分配技术（block preallocation）。如果你想在存储设备上给一个你知道要变大的文件预留空间，ext4文件系统可以为文件分配所有需要用到的块，而不仅仅是那些现在已经用到的块。

6、Reiser文件系统

ReiserFS文件系统只支持回写日志模式——只把索引节点表数据写到日志文件。有两个有意思的特性被引入了ReiserFS文件系统：一个是你可以在线调整已有文件系统的大小；另一个是被称作尾部压缩（tailpacking）的技术，该技术能将一个文件的数据填进另一个文

件的数据块中的空白空间。如果你必须为已有文件系统扩容来容纳更多的数据，在线调整文件系统大小功能非常好用。

7、JFS文件系统

JFS文件系统采用的是有序日志方法，即只在日志中保存索引节点表数据，直到真正的文件数据被写进存储设备时才删除它。JFS文件系统采用基于区段的文件分配，即为每个写入存储设备的文件分配一组块。这样可以减少存储设备上的碎片。

8、XFS文件系统

XFS日志文件系统是另一种最初用于商业Unix系统而如今走进Linux世界的文件系统。

XFS文件系统采用回写模式的日志，在提供了高性能的同时也引入了一定的风险，因为实际数据并未存进日志文件。XFS文件系统还允许在线调整文件系统的大小，这点类似于ReiserFS文件系统，除了XFS文件系统只能扩大不能缩小。

9、就文件系统而言，日志式的另一种选择是一种叫作写时复制（copy-on-write，COW）的技术。COW利用快照兼顾了安全性和性能。如果要修改数据，会使用克隆或可写快照。修改过的数据并不会直接覆盖当前数据，而是被放入文件系统中的另一个位置上。即便是数据修改已经完成，之前的旧数据也不会被重写。

10、ZFS文件系统

ZFS是一个稳定的文件系统，与Resier4、Btrfs和ext4势均力敌。它最大的弱项就是没有使用 GPL许可。

11、Btrf文件系统

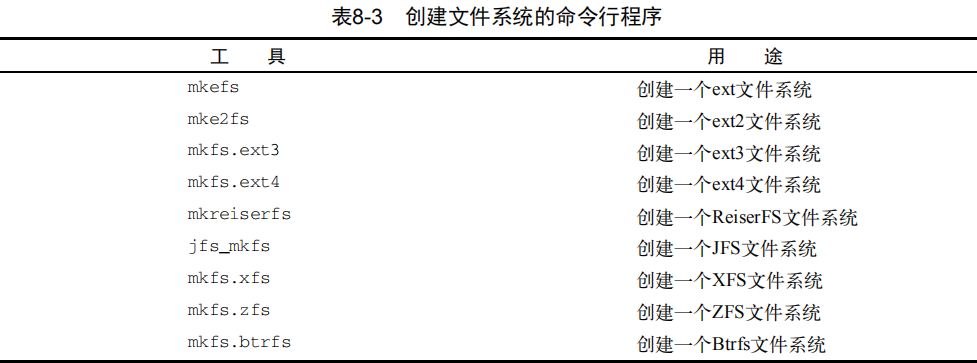
Btrfs文件系统是COW的新人，也被称为B树文件系统。Btrfs在Reiser4的诸多特性的基础上改进了可靠性。另一些开发人员最终也加入了开发过程，帮助Btrfs快速成为了最流行的文件系统。究其原因，则要归于它的稳定性、易用性以及能够动态调整已挂载文件系统的大小。

12、【page 148】fdisk工具用来帮助管理安装在系统上的任何存储设备上的分区。它是个交互式程序，允许你输入命令来逐步完成硬盘分区操作。**sudo fdisk /dev/sdb**

对于老式的IDE驱动器，Linux使用的是/dev/hdx。对于较新的SATA驱动器和SCSI驱动器，Linux使用/dev/sdx。

1. a 设置活动分区标志
2. b 编辑BSD Unix系统用的磁盘标签
3. c 设置DOS兼容标志
4. d 删除分区
5. l 显示可用的分区类型
6. m 显示命令选项
7. n 添加一个新分区
8. o 创建DOS分区表
9. p 显示当前分区表
10. q 退出，不保存更改
11. s 为Sun Unix系统创建一个新磁盘标签
12. t 修改分区的系统ID
13. u 改变使用的存储单位
14. v 验证分区表
15. w 将分区表写入磁盘
16. x 高级功能

13、【page 151】创建文件系统



14、【page 153】fsck命令能够检查和修复大部分类型的Linux文件系统，包括本章早些时候讨论过的ext、ext2、ext3、ext4、ReiserFS、JFS和XFS。该命令的格式是：

fsck options filesystem

只能在未挂载的文件系统上运行fsck命令。

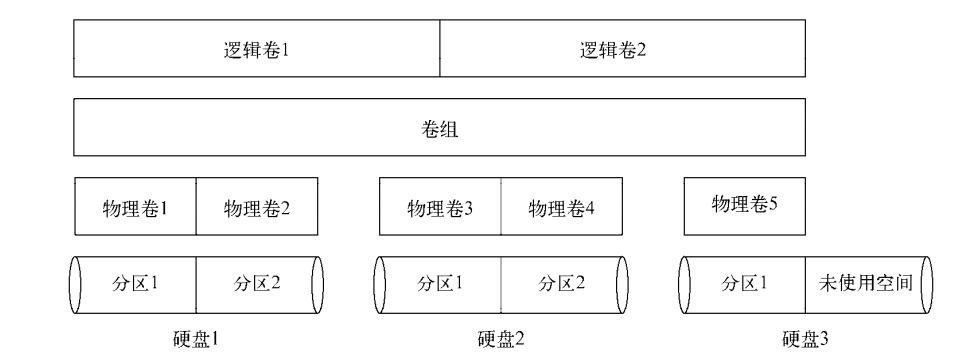
你可以在命令行上列出多个要检查的文件系统。文件系统可以通过设备名、在虚拟目录中的挂载点以及分配给文件系统的唯一UUID值来引用。

fsck命令使用/etc/fstab文件来自动决定正常挂载到系统上的存储设备的文件系统。如果存储设备尚未挂载，你需要用-t命令行选项来指定文件系统类型。

15、逻辑卷管理的核心在于如何处理安装在系统上的硬盘分区。在逻辑卷管理的世界里，硬盘称作物理卷（physical volume，PV）。每个物理卷都会映射到硬盘上特定的物理分区。

多个物理卷集中在一起可以形成一个卷组（volume group，VG）。逻辑卷管理系统将卷组视为一个物理硬盘，但事实上卷组可能是由分布在多个物理硬盘上的多个物理分区组成的。卷组提供了一个创建逻辑分区的平台，而这些逻辑分区则包含了文件系统。

整个结构中的最后一层是逻辑卷（logical volume，LV）。逻辑卷为Linux提供了创建文件系统的分区环境，作用类似于到目前为止我们一直在探讨的Linux中的物理硬盘分区。Linux系统将逻辑卷视为物理分区。



16、LVM主要功能：

（1）在逻辑卷在线的状态下将其复制到另一个设备。这个功能叫作快照。

（2）条带化（striping）可跨多个物理硬盘创建逻辑卷。

（3）镜像是一个实时更新的逻辑卷的完整副本。当你创建镜像逻辑卷时，LVM会将原始逻辑卷同步到镜像副本中。根据原始逻辑卷的大小，这可能需要一些时间才能完成。一旦原始同步完成，LVM会为文件系统的每次写操作执行两次写入——一次写入到主逻辑卷，一次写入到镜像副本。可以想到，这个过程会降低系统的写入性能。就算原始逻辑卷因为某些原因损坏了，你手头也已经有了一个完整的最新副本！

17、【page 158】Linux LVM使用方法

（1）定义物理卷

使用fdisk命令将硬盘上的物理分区转换成Linux LVM使用的物理卷区段。在创建了基本的Linux分区之后，你需要通过t命令改变分区类型，例如得到/dev/sdb1。

通过pvcreate命令用分区来创建实际的物理卷：pvcreate /dev/sdb1

查看创建进度的话，可以使用pvdisplay命令来显示已创建的物理卷列表。

（2）创建卷组

使用vgcreate命令从物理卷中创建一个或多个卷组。vgcreate命令需要一些命令行参数来定义卷组名以及你用来创建卷组的物理卷名：vgcreate Vol1 /dev/sdb1

可用vgdisplay命令看看新创建的卷组的细节。

（3）创建逻辑卷

使用lvcreate命令创建逻辑卷：lvcreate -l 100%FREE -n lvtest Vol1

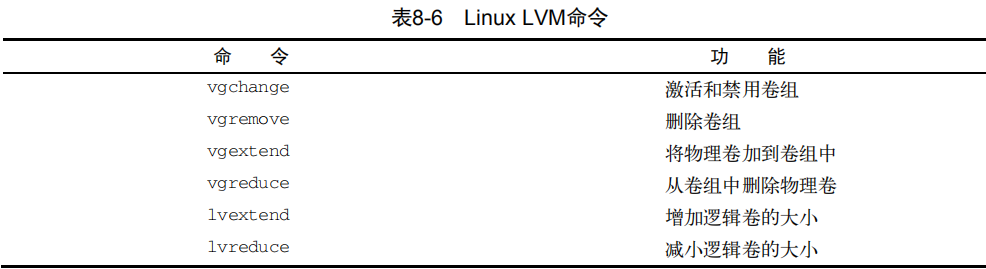
查看你创建的逻辑卷的详细情况，可用lvdisplay命令。

（4）创建文件系统

mkfs.ext4 /dev/Vol1/lvtest

然后用标准Linux mount命令将这个卷挂载到虚拟目录：

mount /dev/Vol1/lvtest /mnt/my\_partition

（5）修改LVM：

# 第10章：使用编辑器

1、readlink -f /usr/bin/vi 查看vi命令连接的vim实体程序。

2、vim编辑器在内存缓冲区中处理数据。只要键入vim命令（或vi，如果这个别名或链接文件存在的话）和要编辑的文件的名字就可以启动vim编辑器：

$ vim myprog.c

3、vim中有用来移动光标的命令。

 h：左移一个字符。

 j：下移一行（文本中的下一行）。

 k：上移一行（文本中的上一行）。

 l：右移一个字符。

 PageDown（或Ctrl+F）：下翻一屏。

 PageUp（或Ctrl+B）：上翻一屏。

 G：移到缓冲区的最后一行。

 num G：移动到缓冲区中的第num行。

 gg：移到缓冲区的第一行

4、在命令行模式下有几个命令可以将缓冲区的数据保存到文件中并退出vim。

q：如果未修改缓冲区数据，退出。

q!：取消所有对缓冲区数据的修改并退出。

w filename：将文件保存到另一个文件中。

wq：将缓冲区数据保存到文件中并退出。

dd：删除当前光标所在的行。

d$：删除当前光标所在位置至行尾的内容。

J：删除当前行的尾部换行符（拼接行）。

u：撤销前一次编辑。

5、复制粘贴：

（1）vim在删除数据时，实际上会将数据保存在单独的一个寄存器中。可以用p命令取回数据。

（2）vim中复制命令是y（代表yank）。可以在y后面使用和d命令相同的第二字符（yw表示复制一个单词，y$表示复制到行尾）。在复制文本后，把光标移动到你想放置文本的地方，输入p命令。复制的文本就会出现在该位置。

（3）进入可视模式，应移动光标到要开始复制的位置，并按下v键。光标所在位置的文本已经被高亮显示了。下一步，移动光标来覆盖你想要复制的文本（甚至可以向下移动几行来复制更多行的文本）。在移动光标时，vim会高亮显示复制区域的文本。在覆盖了要复制的文本后，按y键来激活复制命令。现在寄存器中已经有了要复制的文本，移动光标到你要放置的位置，使用p命令来粘贴。

6、查找替换：

（1）要输入一个查找字符串，就按下斜线（/）键。光标会跑到消息行，然后vim会显示出斜线。在输入你要查找的文本后，按下回车键。

（2）要继续查找同一个单词，按下斜线键，然后按回车键。或者使用n键，表示下一个（next）。

（3）替换命令允许你快速用另一个单词来替换文本中的某个单词。必须进入命令行模式才能使用替换命令。替换命令的格式是：

:s/old/new/

vim编辑器会跳到old第一次出现的地方，并用new来替换。可以对替换命令作一些修改来替换多处文本：

* :s/old/new/g：一行命令替换所有old。
* :n,ms/old/new/g：替换行号n和m之间所有old。
* :%s/old/new/g：替换整个文件中的所有old。
* :%s/old/new/gc：替换整个文件中的所有old，但在每次出现时提示。

7、使用nano打开文件，可以这样：

$ nano myprog.c

在nano编辑器窗口的底部显示了各种命令以及简要的描述。这些命令是nano的控制命令。脱字符（^）表示Ctrl键。因此，^X表示的就是组合键Ctrl+X

# 第11章：构建基本脚本

1、有两种方法可以将命令输出赋给变量：

* 反引号字符（`）：testing='date'
* $()格式：testing=$(date)

2、重定向：

command > outputfile 用新的文件数据覆盖已有文件

command >> outputfile追加数据

command < inputfile 输入重定向

command << inputfile内联输入重定向，无需使用文件进行重定向，只需要在命令行中指定用于输入重定向的数据就可以了

3、wc命令可以对对数据中的文本进行计数。默认情况下，它会输出3个值：

* + 文本的行数
  + 文本的词数
  + 文本的字节数

4、echo $?命令查看退出码，shell中运行的每个命令都使用退出状态码（exit status）告诉shell它已经运行完毕。退出状态码是一个0～255的整数值：

* 0 命令成功结束
* 1 一般性未知错误
* 2 不适合的shell命令
* 126 命令不可执行
* 127 没找到命令
* 128 无效的退出参数
* 128+x 与Linux信号x相关的严重错误
* 130 通过Ctrl+C终止的命令
* 255 正常范围之外的退出状态码

5、exit命令允许你在脚本结束时指定一个退出状态码。

# 第12章：使用结构化命令

1、if-then(-elif)(-else)语句有如下格式。

if command1

then

command set 1

elif command2

then

command set 2

else

command set 3

fi

或者  
if command1; then

command set 1

elif command2; then

command set 2

else

command set 3

fi

在then部分，如果if语句行的命令的退出状态值为0，所有的命令都会被执行；如果if语句行的命令的退出状态不为0，所有的命令都会被跳过。

2、test命令提供了在if-then语句中测试不同条件的途径。如果test命令中列出的条件成立，

test命令就会退出并返回退出状态码0，执行then的语句。如果条件不成立，test命令就会退出并返回非零的退出状态码，不执行then的语句。

if test condition

then

commands

fi

如果不写test命令的condition部分，它会以非零的退出状态码退出，并执行else语句块。

3、bash shell提供了另一种条件测试方法：。

if [ condition ]

then

commands

fi

方括号定义了测试条件。注意，第一个方括号之后和第二个方括号之前必须加上一个空格。

4、条件测试的几种情形：

（1）数值比较

* n1 -eq n2 ： 检查n1是否与n2相等
* n1 -ge n2 ： 检查n1是否大于或等于n2
* n1 -gt n2 ： 检查n1是否大于n2
* n1 -le n2 ： 检查n1是否小于或等于n2
* n1 -lt n2 ： 检查n1是否小于n2
* n1 -ne n2 ： 检查n1是否不等于n2

（2）字符串比较

* str1 = str2 ： 检查str1是否和str2相同
* str1 != str2 ： 检查str1是否和str2不同
* str1 < str2 ： 检查str1是否比str2小
* str1 > str2 ： 检查str1是否比str2大
* -n str1 ： 检查str1的长度是否非0
* -z str1 ： 检查str1的长度是否为0

（3）文件比较

* + -d file ： 检查file是否存在并是一个目录
  + -e file ： 检查file是否存在
  + -f file ： 检查file是否存在并是一个文件
  + -r file ： 检查file是否存在并可读
  + -s file ： 检查file是否存在并非空
  + -w file ： 检查file是否存在并可写
  + -x file ： 检查file是否存在并可执行
  + -O file ： 检查file是否存在并属当前用户所有
  + -G file ： 检查file是否存在并且默认组与当前用户相同
  + file1 -nt file2 ： 检查file1是否比file2新
  + file1 -ot file2 ： 检查file1是否比file2旧

5、复合条件测试

if-then语句允许你使用布尔逻辑来组合测试。有两种布尔运算符可用：

 [ condition1 ] && [ condition2 ]

 [ condition1 ] || [ condition2 ]

6、双括号命令允许你在比较过程中使用高级数学表达式。test命令只能在比较中使用简单的算术操作。

(( expression ))

比如幂运算：(( $val1 \*\* 2 > 90 ))

7、双方括号命令提供了针对字符串比较的高级特性。双方括号命令的格式如下：

[[ expression ]]

双方括号里的expression使用了test命令中采用的标准字符串比较。但它提供了test命

令未提供的另一个特性——模式匹配（pattern matching）。

比如正则表达式：[[ $USER == r\* ]]

8、case命令会采用列表格式来检查单个变量的多个值。

case variable in

pattern1 | pattern2)

commands1;;

pattern3)

commands2;;

\*)

default commands;;

esac

# 第13章：更多的结构化命令

1、for 命令

for var in list

do

commands

done

或者

for var in list; do

commands

done

在list参数中，你需要提供迭代中要用到的一系列值。

bash中C语言风格的for循环的基本格式。

for (( variable assignment ; condition ; iteration process ))

例如：

for (( i=1; i <= 10; i++ ))

do

echo "The next number is $i"

done

2、IFS，叫作内部字段分隔符（internal field separator），IFS环境变量定义了bash shell用作字段分隔符的一系列字符。默认情况下，bash shell会将下列字符当作字段分隔符：

* 空格
* 制表符
* 换行符

（1）如果你想修改IFS的值，使其只能识别换行符，那就必须这么做：

IFS=$'\n'

（2）在改变IFS之前保存原来的IFS值，之后再恢复它。可以这样实现：

IFS.OLD=$IFS

IFS=$'\n'

<在代码中使用新的IFS值>

IFS=$IFS.OLD

（3）如果要指定多个IFS字符，只要将它们在赋值行串起来就行。

IFS=$'\n':;"

3、while命令的格式是：

while test command

do

other commands

done

while命令允许你在while语句行定义多个测试命令。只有最后一个测试命令的退出状态码

会被用来决定什么时候结束循环。

4、until命令要求你指定一个通常返回非零退出状态码的测试命令。只有测试命令的退出状态码不为0，bash shell才会执行循环中列出的命令。一旦测试命令返回了退出状态码0，循环就结束了。和while命令类似，你可以在until命令语句中放入多个测试命令。只有最后一个命令的退出状态码决定了bash shell是否执行已定义的other commands。

until test commands

do

other commands

done

5、通常必须遍历存储在文件中的数据。这要求结合已经讲过的两种技术：

* + 使用嵌套循环
  + 修改IFS环境变量

通过修改IFS环境变量，就能强制for命令将文件中的每行都当成单独的一个条目来处理，即便数据中有空格也是如此。一旦从文件中提取出了单独的行，可能需要再次利用循环来提取行中的数据。

6、有两个命令能帮我们控制循环内部的情况：

* break命令
* continue命令

（1）break命令接受单个命令行参数值：

break n

其中n指定了要跳出的循环层级。默认情况下，n为1，表明跳出的是当前的循环。如果你将n设为2，break命令就会停止下一级的外部循环。

（2）continue命令也允许通过命令行参数指定要继续执行哪一级循环：

continue n

其中n定义了要继续的循环层级。

7、输出控制

输出重定向：done > file

输出到管道：done > cmd

# 第14章：处理用户输入

1、bash shell会将一些称为位置参数（positional parameter）的特殊变量分配给输入到命令行中的所有参数。位置参数变量是标准的数字：

* $0是程序名，
* $1是第一个参数，$2是第二个参数，依次类推，直到第九个参数$9，空格分隔。

2、basename命令会返回不包含路径的脚本名。通常使用$0时可以用到。

3、特殊变量$#含有脚本运行时携带的命令行参数的个数。

${!#}在有输入参数的事后反悔最后一个参数，否则返回脚本名（$0）

4、$\*和$@变量可以用来轻松访问所有的参数。这两个变量都能够在单个变量中存储所有的命令行参数。

$\*变量会将命令行上提供的所有参数当作一个单词保存。这个单词包含了命令行中出现的每一个参数值。基本上$\*变量会将这些参数视为一个整体，而不是多个个体。

$@变量会将命令行上提供的所有参数当作同一字符串中的多个独立的单词。这样你就能够遍历所有的参数值，得到每个参数。这通常通过for命令完成。

5、shift命令能够用来操作命令行参数。在使用shift命令时，默认情况下它会将每个参数变量向左移动一个位置。所以，变量$3的值会移到$2中，变量$2的值会移到$1中，而变量$1的值则会被删除。如果某个参数被移出，它的值就被丢弃了，无法再恢复。

6、shell会用双破折线（--）来表明选项列表结束。在双破折线之后，脚本就可以放心地将剩下的命令行参数当作参数，而不是选项来处理了。

如： ./test16.sh -c -a -b -- test1 test2 test3

7、getopt命令可以接受一系列任意形式的命令行选项和参数，并自动将它们转换成适当的格式。它的命令格式如下：

getopt optstring parameters

首先，在optstring中列出你要在脚本中用到的每个命令行选项字母。然后，在每个需要参数值的选项字母后加一个冒号。getopt命令会基于你定义的optstring解析提供的参数。

例如：getopt ab:cd -a -b test1 -cd test2 test3（b选项需要参数值）

等价于-a -b test1 -c -d -- test2 test3

如果指定了一个不在optstring中的选项，默认情况下，getopt命令会产生一条错误消息。如果想忽略这条错误消息，可以在命令后加-q选项。

8、set命令替换原输入参数的选项之一是双破折线（--），它会将命令行参数替换成set命令的命令行值。

然后，该方法会将原始脚本的命令行参数传给getopt命令，之后再将getopt命令的输出传给set命令，用getopt格式化后的命令行参数来替换原始的命令行参数：

set -- $(getopt -q ab:cd "$@")

9、与getopt不同，前者将命令行上选项和参数处理后只生成一个输出，而getopts命令每次调用时，一次只处理命令行上检测到的一个参数。处理完所有的参数后，它会退出并返回一个大于0的退出状态码。getopts命令的格式如下：

getopts optstring variable

（1）要去掉错误消息的话，可以在optstring之前加一个冒号。getopts命令将当前参数保存在命令行中定义的variable中。

（2）如果选项需要跟一个参数值，OPTARG环境变量就会保存这个值。OPTIND环境变量保存了参数列表中getopts正在处理的参数位置。这样你就能在处理完选项之后继续处理其他命令行参数了。

10、常用的Linux命令选项

* -a 显示所有对象
* -c 生成一个计数
* -d 指定一个目录
* -e 扩展一个对象
* -f 指定读入数据的文件
* -h 显示命令的帮助信息
* -i 忽略文本大小写
* -l 产生输出的长格式版本
* -n 使用非交互模式（批处理）
* -o 将所有输出重定向到的指定的输出文件
* -q 以安静模式运行
* -r 递归地处理目录和文件
* -s 以安静模式运行
* -v 生成详细输出
* -x 排除某个对象
* -y 对所有问题回答yes

11、read命令从标准输入（键盘）或另一个文件描述符中接受输入。在收到输入后，read命令会将数据放进一个变量。

read命令包含了-p选项，允许你直接在read命令行指定提示符：

read -p "Please enter your age: " age

（1）输入的每个数据值都会分配给变量列表中的下一个变量。如果变量数量不够，剩下的数据就全部分配给最后一个变量。

（2）不指定变量，read命令会将它收到的任何数据都放进特殊环境变量REPLY中。

（3）用-t选项来指定一个计时器。-t选项指定了read命令等待输入的秒数。当计时器过期后，read命令会返回一个非零退出状态码。

（4）-s选项可以避免在read命令中输入的数据出现在显示器上（实际上，数据会被显示，只是read命令会将文本颜色设成跟背景色一样）。

（5）将文件中的数据传给read命令：

cat test | while read line

do

commands

done

# 第15章：呈现数据

1、bash shell保留了前三个文 件描述符（0标准输入、1标准输出和2标准错误）

（1）STDIN文件描述符代表shell的标准输入。对终端界面来说，标准输入是键盘。

在使用输入重定向符号（<）时，Linux会用重定向指定的文件来替换标准输入文件描述符。它会读取文件并提取数据，就如同它是键盘上键入的。

（2）STDOUT文件描述符代表shell的标准输出。在终端界面上，标准输出就是终端显示器。shell的所有输出（包括shell中运行的程序和脚本）会被定向到标准输出中，也就是显示器。

默认情况下，大多数bash命令会将输出导向STDOUT文件描述符。

（3）STDERR文件描述符代表shell的标准错误输出。shell或shell中运行的程序和脚本出错时生成的错误消息都会发送到这个位置。

默认情况下，STDERR文件描述符会和STDOUT文件描述符指向同样的地方（显示器）。但STDERR并不会随着STDOUT的重定向而发生改变。

2、当使用&>符时，命令生成的所有输出都会发送到同一位置，包括输出和错误。为了避免错误信息散落在输出文件中，相较于标准输出，bashshell自动赋予了错误消息更高的优先级。

3、输出临时重定向：如果有意在脚本中生成错误消息，可以将单独的一行输出重定向到STDERR。

echo "This is an error message" >&2

4、输出永久重定向：如果脚本中有大量数据需要重定向，那重定向每个echo语句就会很烦琐。取而代之，你可以用exec命令告诉shell在脚本执行期间重定向某个特定文件描述符。

exec 1>testout

exec命令会启动一个新shell并将STDOUT文件描述符重定向到文件。脚本中发给STDOUT的所有输出会被重定向到文件。

5、exec命令允许你将STDIN重定向到Linux系统上的文件中：

exec 0< testfile

6、在shell中最多可以有9个打开的文件描述符。其他6个从3~8的文件描述符均可用作输入或输出重定向。

（1）可以用exec命令来给输出分配文件描述符。和标准的文件描述符一样，一旦将另一个文件描述符分配给一个文件，这个重定向就会一直有效，直到你重新分配。

exec 3>test13out 或者 exec 3>>test13out

（2）文件描述符临时修改后再恢复的方法，输出：

exec 3>&1; exec 1>test14out

comands

exec 1>&3

输入：

exec 6<&0; exec 0< testfile

comands

exec 0<&6

（3）对同一个文件进行数据读写，shell会维护一个内部指针，指明在文件中的当前位置。任何读或写都会从文件指针上次的位置开始。

exec 3<> testfile

read line <&3

echo "This is a test line" >&3

（4）要关闭文件描述符，将它重定向到特殊符号&-。脚本中看起来如下：

exec 3>&-

一旦关闭了文件描述符，就不能在脚本中向它写入任何数据，否则shell会生成错误消息。

7、lsof命令会列出整个Linux系统打开的所有文件描述符。lsof的默认输出中有7列信息：

* COMMAND 正在运行的命令名的前9个字符
* PID 进程的PID
* USER 进程属主的登录名
* FD 文件描述符号以及访问类型（r代表读，w代表写，u代表读写）
* TYPE 文件的类型（CHR代表字符型，BLK代表块型，DIR代表目录，REG代表常规文件）
* DEVICE 设备的设备号（主设备号和从设备号）
* SIZE 如果有的话，表示文件的大小
* NODE 本地文件的节点号
* NAME 文件名

（1）选项-p允许指定进程ID（PID），

（2）选项-d允许指定要显示的文件描述符编号。

（3）-a选项用来对其他两个选项的结果执行布尔AND运算。

8、要想知道进程的当前PID，可以用特殊环境变量$$（shell会将它设为当前PID）。

9、阻止命令输出：在Linux系统上null文件的标准位置是/dev/null。你重定向到该位置的任何数据都会被丢掉，不会显示。

例如：2> /dev/null

10、Linux使用/tmp目录来存放临时文件。大多数Linux发行版配置了系统在启动时自动删除/tmp目录的所有文件。系统上的任何用户账户都有权限在读写/tmp目录中的文件。

mktemp命令可以在/tmp目录中创建一个唯一的临时文件。shell会创建这个文件，但不用默认的umask值。它会将文件的读和写权限分配给文件的属主，并将你设成文件的属主。一旦创建了文件，你就在脚本中有了完整的读写权限，但其他人没法访问它（当然，root用户除外）。

（1）要用mktemp命令在本地目录中创建一个本地临时文件，你只要指定一个文件名模板就行了。模板可以包含任意文本文件名，在文件名末尾加上6个X就行了：

$ mktemp testing.XXXXXX

例如：

tempfile=$(mktemp test19.XXXXXX)

echo "This script writes to temp file $tempfile"

（2）-t选项会强制mktemp命令来在系统的临时目录来创建该文件。在用这个特性时，mktemp命令会返回用来创建临时文件的全路径，而不是只有文件名。

由于mktemp命令返回了全路径名，你可以在Linux系统上的任何目录下引用该临时文件，不管临时目录在哪里。

（3）-d选项告诉mktemp命令来创建一个临时目录而不是临时文件。

11、tee命令相当于管道的一个T型接头。它将从STDIN过来的数据同时发往两处。一处是

STDOUT，另一处是tee命令行所指定的文件名：

tee filename

默认情况下，tee命令会在每次使用时覆盖输出文件内容。如果你想将数据追加到文件中，必须用-a选项。

# 第16章：控制脚本

1、最常见的Linux系统信号：

* 1 SIGHUP 挂起进程
* 2 SIGINT 终止进程
* 3 SIGQUIT 停止进程
* 9 SIGKILL 无条件终止进程
* 15 SIGTERM 尽可能终止进程
* 17 SIGSTOP 无条件停止进程，但不终止进程
* 18 SIGTSTP 停止或暂停进程，但不终止进程
* 19 SIGCONT 继续运行停止的进程

（1）默认情况下，bash shell会忽略收到的任何SIGQUIT (3)和SIGTERM (5)信号（正因为这样，交互式shell才不会被意外终止）。

（2）如果bash shell收到了SIGHUP信号（关闭会话），比如当你要离开一个交互式shell，它就会退出。但在退出之前，它会将SIGHUP信号传给所有由该shell所启动的进程。

（3）通过SIGINT信号（Ctrl+C组合键），可以中断shell。Linux内核会停止为shell分配CPU处理时间。这种情况发生时，shell会将SIGINT信号传给所有由它所启动的进程，以此告知出现的状况。

（4）Ctrl+Z组合键会生成一个SIGTSTP信号，停止shell中运行的任何进程。停止（stopping）进程跟终止（terminating）进程不同：停止进程会让程序继续保留在内存中，并能从上次停止的位置继续运行。

2、trap命令允许你来指定shell脚本要监看并从shell中拦截的Linux信号。如果脚本收到了trap命令中列出的信号，该信号不再由shell处理，而是交由本地处理。

trap commands signals

（1）要捕获shell脚本的退出，只要在trap命令后加上EXIT信号就行。

trap commands EXIT

（2）要想在脚本中的不同位置进行不同的捕获处理，只需重新使用带有新选项的trap命令。

（3）删除捕获在trap命令与希望恢复默认行为的信号列表之间加上两个破折号。

trap -- signals

移除信号捕获后，脚本按照默认行为来处理SIGINT信号

也可以在trap命令后使用单破折号来恢复信号的默认行为。单破折号和双破折号都可以

正常发挥作用。

3、nohup命令运行了另外一个命令来阻断所有发送给该进程的SIGHUP信号。这会在退出终端会话时阻止进程退出。

$ nohup ./test1.sh &

nohup命令会自动将STDOUT和STDERR的消息重定向到一个名为nohup.out的文件中。

4、jobs命令允许查看shell当前正在处理的作业

* + -l 列出进程的PID以及作业号
  + -n 只列出上次shell发出的通知后改变了状态的作业
  + -p 只列出作业的PID
  + -r 只列出运行中的作业
  + -s 只列出已停止的作业

带加号的作业会被当做默认作业。在使用作业控制命令时，如果未在命令行指定任何作业号，该作业会被当成作业控制命令的操作对象。

当前的默认作业完成处理后，带减号的作业成为下一个默认作业。任何时候都只有一个带加号的作业和一个带减号的作业，不管shell中有多少个正在运行的作业。

5、要以后台模式重启一个作业，可用bg命令加上作业号。

要以前台模式重启作业，可用带有作业号的fg命令。

6、调度优先级是个整数值，从-20（最高优先级）到+19（最低优先级）。默认情况下，bash shell以优先级0来启动所有进程。

7、nice命令允许你设置命令启动时的调度优先级。要让命令以更低的优先级运行，只要用nice的-n命令行来指定新的优先级级别。nice命令阻止普通系统用户来提高命令的优先级。

nice命令的-n选项并不是必须的，只需要在破折号后面跟上优先级就行了。

nice -10 ./test4.sh > test4.out &

8、renice命令允许你指定运行中的进程的PID来改变它的优先级。

* 只能对属于你的进程执行renice；
* 只能通过renice降低进程的优先级；
* root用户可以通过renice来任意调整进程的优先级。

如果想完全控制运行进程，必须以root账户身份登录或使用sudo命令。

9、at命令允许指定Linux系统何时运行脚本。at命令会将作业提交到队列中，指定shell何时运行该作业。at的守护进程atd会以后台模式运行，检查作业队列来运行作业。大多数Linux发行版会在启动时运行此守护进程。

（1）atd守护进程会检查系统上的一个特殊目录（通常/var/spool/at）来获取用at命令提交的作业。默认情况下，atd守护进程会每60秒检查一下这个目录。有作业时，atd守护进程会检查作业设置运行的时间。如果时间跟当前时间匹配，atd守护进程就会运行此作业。

at [-f filename] time

（2）在你使用at命令时，该作业会被提交到作业队列（job queue）。作业队列会保存通过at命令提交的待处理的作业。针对不同优先级，存在26种不同的作业队列。作业队列通常用小写字母a~z和大写字母A~Z来指代。作业队列的字母排序越高，作业运行的优先级就越低（更高的nice值）。默认情况下，at的作业会被提交到a作业队列。如果想以更高优先级运行作业，可以用-q参数指定不同的队列字母。

（3）当作业在Linux系统上运行时，显示器并不会关联到该作业。取而代之的是，Linux系统会将提交该作业的用户的电子邮件地址作为STDOUT和STDERR。果你的系统中没有安装sendmail，那就无法获得任何输出！因此在使用at命令时，最好在脚本中对STDOUT和STDERR进行重定向。如果不想在at命令中使用邮件或重定向，最好加上-M选项来屏蔽作业产生的输出信息。

（4）atq命令可以查看系统中有哪些作业在等待。

（5）atrm命令用来删除等待中的作业。

10、Linux系统使用cron程序来安排要定期执行的作业。cron程序会在后台运行并检查一个特殊的表（被称作cron时间表），以获知已安排执行的作业。

时间格式：min hour dayofmonth month dayofweek command

11、crontab命令可以用-l选项列出已有的cron时间表。

要为cron时间表添加条目，可以用-e选项。

12、如果anacron知道某个作业错过了执行时间，它会尽快运行该作业。这意味着如果Linux系统关机了几天，当它再次开机时，原定在关机期间运行的作业会自动运行。

anacron时间表：period delay identifier command

# 第17章：创建函数

1、有两种格式可以用来在bash shell脚本中创建函数：

function name {

commands

}

或者

name() {

commands

}

2、有3种不同的方法来为函数生成退出状态码：

（1）默认情况下，函数的退出状态码是函数中最后一条命令返回的退出状态码。

（2）使用return命令来退出函数并返回特定的退出状态码（0-255）。允许指定一个整数值来定义函数的退出状态码，从而提供了一种简单的途径来编程设定函数退出状态码。

如果在用$?变量提取函数返回值之前执行了其他命令，函数的返回值就会丢失。记住，$?变量会返回执行的最后一条命令的退出状态码。

（3）对函数的输出保存到变量（可以是浮点型、字符串等）：

function function1{

read -p "Enter a value: " value

echo $[ $value \* 2 ]

}

result=$(function1)

3、函数变量：

（1）入参：函数名会在$0变量中定义，函数命令行上的任何参数都会通过$1、$2等定义。也可以用特殊变量$#来判断传给函数的参数数目。

在脚本中指定函数时，必须将参数和函数放在同一行，像这样：

value=$(func1 $1 $2)

（2）局部：函数内部使用的任何变量都可以被声明成局部变量：

local temp

4、如果你试图将该数组变量作为函数参数，函数只会取数组变量的第一个值。你必须将该数组变量的值分解成单个的值，然后将这些值作为函数参数使用。在函数内部，可以将所有的参数重新组合成一个新的变量。

function testit {

local newarray

newarray=(;'echo "$@"')

}

myarray=(1 2 3 4 5)

testit ${myarray[\*]}

5、从函数里向shell脚本传回数组变量也用类似的方法。函数用echo语句来按正确顺序输出单个数组值，然后脚本再将它们重新放进一个新的数组变量中。

function arraydblr {

echo ${newarray[\*]}

}

6、局部函数变量的一个特性是自成体系。shell里的函数支持递归调用。

7、在命令行上创建函数有两种方法

（1）单行方式定义函数

function divem { echo $[ $1 / $2 ]; }

（2）多行方式来定义函数

$ function multem {

> echo $[ $1 \* $2 ]

> }

8、在.bashrc 文件中定义函数有两种方法

（1）可以直接在主目录下的.bashrc文件中定义函数。

（2）用source命令（或者它的别名点操作符）将库文件中的函数添加到.bashrc脚本中。

9、shtool库提供了一些简单的shell脚本函数，可以用来完成日常的shell功能，例如处理临时文件和目录或者格式化输出显示。

ftp://ftp.gnu.org/gnu/shtool/shtool-2.0.8.tar.gz

$ tar -zxvf shtool-2.0.8.tar.gz

$ ./confifgure

$ make

$ sudo make install

shtool库函数:

* + Arx 创建归档文件（包含一些扩展功能）
  + Echo 显示字符串，并提供了一些扩展构件
  + fixperm 改变目录树中的文件权限
  + install 安装脚本或文件
  + mdate 显示文件或目录的修改时间
  + mkdir 创建一个或更多目录
  + Mkln 使用相对路径创建链接
  + mkshadow 创建一棵阴影树
  + move 带有替换功能的文件移动
  + Path 处理程序路径
  + platform 显示平台标识
  + Prop 显示一个带有动画效果的进度条
  + rotate 转置日志文件
  + Scpp 共享的C预处理器
  + Slo 根据库的类别，分离链接器选项
  + Subst 使用sed的替换操作
  + Table 以表格的形式显示由字段分隔（field-separated）的数据
  + tarball 从文件和目录中创建tar文件
  + version 创建版本信息文件

# 第18章：图形化桌面环境中的脚本编程

1、clear命令用当前终端会话的terminfo数据（参见第2章）来清理出现在屏幕上的文本。

2、select命令只需要一条命令就可以创建出菜单，然后获取输入的答案并自动处理。select

命令的格式如下。

select variable in list

do

commands

done

3、【page 384】dialog包最早是由Savio Lam创建的一个小巧的工具，能够用ANSI转义控制字符在文本环境中创建标准的窗口对话框。你可以轻而易举地将这些对话框融入自己的shell脚本中，借此与用户进行交互。

要在命令行上指定某个特定的部件，需使用双破折线格式:

dialog --widget parameters

# 第19章：初识sed和gawk

1、sed编辑器被称作流编辑器（stream editor），和普通的交互式文本编辑器恰好相反。流编辑器则会在编辑器处理数据之前基于预先提供的一组规则来编辑数据流。

(1) 一次从输入中读取一行数据。

(2) 根据所提供的编辑器命令匹配数据。

(3) 按照命令修改流中的数据。

(4) 将新的数据输出到STDOUT。

在流编辑器将所有命令与一行数据匹配完毕后，它会读取下一行数据并重复这个过程。在流编辑器处理完流中的所有数据行后，它就会终止。由于命令是按顺序逐行给出的，sed编辑器只需对数据流进行一遍处理就可以完成编辑操作，sed命令的格式如下。

sed options script file

* -e script 在处理输入时，将script中指定的命令添加到已有的命令中
* -f file 在处理输入时，将file中指定的命令添加到已有的命令中
* -n 不产生命令输出，使用print命令来完成输出

（1）默认情况下，sed编辑器会将指定的命令应用到STDIN输入流上。这样你可以直接将数据通过管道输入sed编辑器处理。

echo "This is a test" | sed 's/test/big test/'

sed 's/dog/cat/' data1.txt

sed编辑器并不会修改文本文件的数据。它只会将修改后的数据发送到STDOUT。

要在sed命令行上执行多个命令时，只要用-e选项就可以了。命令之间必须用分号隔开，并且在命令末尾和分号之间不能有空格。

2、在gawk编程语言中，你可以做下面的事情：

*  定义变量来保存数据；
*  使用算术和字符串操作符来处理数据；
*  使用结构化编程概念（比如if-then语句和循环）来为数据处理增加处理逻辑；
*  通过提取数据文件中的数据元素，将其重新排列或格式化，生成格式化报告。

gawk程序的基本格式如下：

gawk options program file

-F fs 指定行中划分数据字段的字段分隔符

-f file 从指定的文件中读取程序

-v var=value 定义gawk程序中的一个变量及其默认值

-mf N 指定要处理的数据文件中的最大字段数

-mr N 指定数据文件中的最大数据行数

-W keyword 指定gawk的兼容模式或警告等级

（1）gawk程序脚本用一对花括号来定义。你必须将脚本命令放到两个花括号（{}）中。

gawk '{print "Hello World!"}'

（2）gawk编程语言允许你将多条命令组合成一个正常的程序。要在命令行上的程序脚本中使用多条命令，只要在命令之间放个分号即可。

$ echo "My name is Rich" | gawk '{$4="Christine"; print $0}'

（3）允许将程序存储到文件中，然后再在命令行中引用：

gawk -F: -f script2.gawk /etc/passwd

（4）gawk还允许指定程序脚本何时运行。BEGIN 关键字就是用来做这个的。它会强制gawk在读取数据前执行BEGIN关键字后指定的程序脚本。

gawk 'BEGIN {print "Hello World!"}

Hello'

END关键字允许你指定一个程序脚本，gawk会在读完数据后执行它。

gawk 'BEGIN {print "The data3 File Contents:"}

Hello'

3、sed基础：

（1）替换标记：s/pattern/replacement/flags

flags：

*  数字，表明新文本将替换第几处模式匹配的地方；
*  g，表明新文本将会替换所有匹配的文本；
*  p，表明原先行的内容要打印出来；
*  w file，将替换的结果写到文件中。

（2）想将命令作用于特定行或某些行，则必须用行寻址：

[address]command 或者

address {

command1

command2

command3

}

行地址区间：

$ sed '2,3s/dog/cat/' data1.txt

中从某行开始的所有行

$ sed '2,$s/dog/cat/' data1.txt

指定文本模式（感觉像关键字或者正则表达式）来过滤出命令要作用的行。格式如下：

/pattern/command

例如：$ sed '/Samantha/s/bash/csh/' /etc/passwd

（3）如果需要在单行上执行多条命令，可以用花括号将多条命令组合在一起：

$ sed '2{

> s/fox/elephant/

> s/dog/cat/

> }' data1.txt

（4）删除命令d名副其实，它会删除匹配指定寻址模式的所有行。使用该命令时要特别小心，如果你忘记加入寻址模式的话，流中的所有文本行都会被删除。

sed '/number 1/d' data6.txt

（5）插入（insert）命令（i）会在指定行前增加一个新行，附加（append）命令（a）会在指定行后增加一个新行：

sed '[address]command\

new line'

echo "Test Line 2" | sed 'i\Test Line 1'

echo "Test Line 2" | sed 'a\Test Line 1'

（6）修改（change）命令允许修改数据流中整行文本的内容。它跟插入和附加命令的工作机制一样，你必须在sed命令中单独指定新行。

$ sed '/number 3/c\

> This is a changed line of text.' data6.txt

（7）转换（transform）命令（y）是唯一可以处理单个字符的sed编辑器命令。转换命令格式如下：

[address]y/inchars/outchars/

转换命令会对inchars和outchars值进行一对一的映射。inchars中的第一个字符会被转换为outchars中的第一个字符，第二个字符会被转换成outchars中的第二个字符。这个映射过程会一直持续到处理完指定字符。如果inchars和outchars的长度不同，则sed编辑器会产生一 条错误消息。

$ sed 'y/123/789/' data8.txt

（8）有3个命令也能用来打印数据流中的信息：

* p命令用来打印文本行；
* 等号（=）命令用来打印行号；
* l（小写的L）命令用来列出行。

$ sed -n '/number 3/p' data6.txt

$ sed '=' data1.txt

$ sed -n 'l' data9.txt

（9）w命令用来向文件写入行。该命令的格式如下：

[address]w filename

读取（read）命令（r）允许你将一个独立文件中的数据插入到数据流中。

[address]r filename

# 第20章：正则表达式

1、正则表达式是你所定义的模式模板（pattern template），Linux工具可以用它来过滤文本。Linux工具（比如sed编辑器或gawk程序）能够在处理数据时使用正则表达式对数据进行模式匹配。如果数据匹配模式，它就会被接受并进一步处理；如果数据不匹配模式，它就会被滤掉。

2、正则表达式是通过正则表达式引擎（regular expression engine）实现的。正则表达式引擎是一套底层软件，负责解释正则表达式模式并使用这些模式进行文本匹配。

* POSIX基础正则表达式（basic regular expression，BRE）引擎
* POSIX扩展正则表达式（extended regular expression，ERE）引擎

3、正则表达式识别的特殊字符包括：

**.\*[]^${}\+?|()**

（1）如果要用某个特殊字符作为文本字符，就必须转义（字符\）。

（2）脱字符（^）定义从数据流中文本行的行首开始的模式。如果模式出现在行首之外的位置，正则表达式模式则无法匹配。用脱字符必须将它放在正则表达式中指定的模式前面。

sed -n '/^ this/p' data3

（3）特殊字符美元符（$）定义了行尾锚点。将这个特殊字符放在文本模式之后来指明数据行必须以该文本模式结尾。

echo "This is a good book" | sed -n '/book$/p'

（4）特殊字符点号（.）用来匹配除换行符之外的任意单个字符。它必须匹配一个字符，如果在点号字符的位置没有字符，那么模式就不成立。

（5）使用方括号[]来定义一个字符组。方括号中包含所有你希望出现在该字符组中的字符。然后你可以在模式中使用整个组，就跟使用其他通配符一样。

例如：echo "Yes" | sed -n '/[Yy]es/p'

（6）使用（^[]）通过排除型字符组，正则表达式模式会匹配c或h之外的任何字符以及文本模式。

（7）可以用单破折线符号在字符组中表示字符区间。只需要指定区间的第一个字符、单破折线以及区间的最后一个字符就行了。根据Linux系统采用的字符集（参见第2章），正则表达式会包括此区间内的任意字符。

$ sed -n '/^[0-9][c-fo-p]$/p' data8

（8）特殊的字符组:

* [[:alpha:]] 匹配任意字母字符，不管是大写还是小写
* [[:alnum:]] 匹配任意字母数字字符0~9、A~Z或a~z
* [[:blank:]] 匹配空格或制表符
* [[:digit:]] 匹配0~9之间的数字
* [[:lower:]] 匹配小写字母字符a~z
* [[:print:]] 匹配任意可打印字符
* [[:punct:]] 匹配标点符号
* [[:space:]] 匹配任意空白字符：空格、制表符、NL、FF、VT和CR
* [[:upper:]] 匹配任意大写字母字符A~Z

（9）在字符后面放置星号（\*）表明该字符必须在匹配模式的文本中出现0次或多次。

4、ERE：gawk程序能够识别ERE模式，但sed编辑器不能。

（1）问号（?）类似于星号，不过问号表明前面的字符可以出现0次或1次，但只限于此。它不会匹配多次出现的字符。

（2）加号（+）是类似于星号的另一个模式符号，但跟问号也有不同。加号表明前面的字符可以出现1次或多次，但必须至少出现1次。如果该字符没有出现，那么模式就不会匹配。

（3）花括号（{}）允许你为可重复的正则表达式指定一个上限。这通常称为间隔（interval）。

可以用两种格式来指定区间。

* m：正则表达式准确出现m次。
* m, n：正则表达式至少出现m次，至多n次。

默认情况下，gawk程序不会识别正则表达式间隔。必须指定gawk程序的--re- interval命令行选项才能识别正则表达式间隔。

echo "beet" | gawk --re-interval '/be{1}t/{print $0}'

（4）管道符号（|）允许你在检查数据流时，用逻辑OR方式指定正则表达式引擎要用的两个或多个模式。如果任何一个模式匹配了数据流文本，文本就通过测试。如果没有模式匹配，则数据流文本匹配失败。使用管道符号的格式如下：

expr1|expr2|...

（5）正则表达式模式也可以用圆括号（()）进行分组。当你将正则表达式模式分组时，该组会被视为一个标准字符。可以像对普通字符一样给该组使用特殊字符。

echo "Saturday" | gawk '/Sat(urday)?/{print $0}'

# 第21章：sed进阶

1、小写的n命令会告诉sed编辑器移动到数据流中的下一文本行，而不用重新回到命令的最开始再执行一遍。

单行next命令会将数据流中的下一文本行移动到sed编辑器的工作空间（称为模式空间）。多行版本的next命令（用大写N）会将下一文本行添加到模式空间中已有的文本后。

2、模式空间（pattern space）是一块活跃的缓冲区，在sed编辑器执行命令时它会保存待检查的文本。

sed编辑器有另一块称作保持空间（hold space）的缓冲区域。在处理模式空间中的某些行时，可以用保持空间来临时保存一些行。有5条命令可用来操作保持空间：

h 将模式空间复制到保持空间

H 将模式空间附加到保持空间

g 将保持空间复制到模式空间

G 将保持空间附加到模式空间

x 交换模式空间和保持空间的内容

通常，在使用h或H命令将字符串移动到保持空间后，最终还要用g、G或x命令将保存的字符串移回模式空间。

3、感叹号命令（!）用来排除（negate）命令，也就是让原本会起作用的命令不起作用。

sed -n '/header/!p' data2.txt #打印没有header的行

sed编辑器无法处理数据流中最后一行文本，因为之后再没有其他行了。可以用感叹号来解决这个问题。

sed 'N;

4、sed编辑器提供了一种方法，可以基于地址、地址模式或地址区间排除一整块命令。这允许你只对数据流中的特定行执行一组命令。

分支（branch）命令b的格式如下：

[address]b [label]

address参数决定了哪些行的数据会触发分支命令。label参数定义了要跳转到的位置。如

果没有加label参数，跳转命令会跳转到脚本的结尾。

5、类似于分支命令，测试（test）命令（t）也可以用来改变sed编辑器脚本的执行流程。测试命令会根据替换命令的结果跳转到某个标签，而不是根据地址进行跳转。

如果替换命令成功匹配并替换了一个模式，测试命令就会跳转到指定的标签。如果替换命令未能匹配指定的模式，测试命令就不会跳转。

[address]t [label]

6、&符号可以用来代表替换命令中的匹配的模式。不管模式匹配的是什么样的文本，你都可以在替代模式中使用&符号来使用这段文本。这样就可以操作模式所匹配到的任何单词了。

$ echo "The cat sleeps in his hat." | sed 's/.at/"&"/g'

&符号会提取匹配替换命令中指定模式的整个字符串。sed编辑器用圆括号来定义替换模式中的子模式。你可以在替代模式中使用特殊字符来引用每个子模式。替代字符由反斜线和数字组成。数字表明子模式的位置。sed编辑器会给第一个子模式分配字符\1，给第二个子模式分配字符\2，依此类推。

# 第22章：gawk进阶

1、gawk数据字段和记录变量

FIELDWIDTHS 由空格分隔的一列数字，定义了每个数据字段确切宽度

FS 输入字段分隔符

RS 输入记录分隔符

OFS 输出字段分隔符

ORS 输出记录分隔符

其他内建变量

ARGC 当前命令行参数个数

ARGIND 当前文件在ARGV中的位置

ARGV 包含命令行参数的数组

CONVFMT 数字的转换格式（参见printf语句），默认值为%.6 g

ENVIRON 当前shell环境变量及其值组成的关联数组

ERRNO 当读取或关闭输入文件发生错误时的系统错误号

FILENAME 用作gawk输入数据的数据文件的文件名

FNR 当前数据文件中的数据行数

IGNORECASE 设成非零值时，忽略gawk命令中出现的字符串的字符大小写

NF 数据文件中的字段总数

NR 已处理的输入记录数

OFMT 数字的输出格式，默认值为%.6 g

RLENGTH 由match函数所匹配的子字符串的长度

RSTART 由match函数所匹配的子字符串的起始位置

2、可以用标准赋值语句来定义数组变量。数组变量赋值的格式如下：

var[index] = element

（1）如果要在gawk中遍历一个关联数组，可以用for语句的一种特殊形式。

for (var in array)

{

statements

}

（2）从关联数组中删除数组索引要用一个特殊的命令。

delete array[index]

3、gawk结构化语句

（1）if语句

if (condition) statement1

（2）while语句

while (condition)

{

statements

} 以及

do

{

statements

} while (condition)

（3）for语句

for( variable assignment; condition; iteration process)

4、格式化打印：

printf "format string", var1, var2 . . .

格式化指定符采用如下格式：

%[modifier]control-letter

控制字母：

c 将一个数作为ASCII字符显示

d 显示一个整数值

i 显示一个整数值（跟d一样）

e 用科学计数法显示一个数

f 显示一个浮点值

g 用科学计数法或浮点数显示（选择较短的格式）

o 显示一个八进制值

s 显示一个文本字符串

x 显示一个十六进制值

X 显示一个十六进制值，但用大写字母A~F

除了控制字母外，还有3种修饰符可以用来进一步控制输出。

* width：指定了输出字段最小宽度的数字值。如果输出短于这个值，printf会将文本右对齐，并用空格进行填充。如果输出比指定的宽度还要长，则按照实际的长度输出。
* prec：这是一个数字值，指定了浮点数中小数点后面位数，或者文本字符串中显示的最 大字符数。
* -（减号）：指明在向格式化空间中放入数据时采用左对齐而不是右对齐。

5、gawk函数

（1）数学函数

atan2(x, y) *x*/*y*的反正切，*x*和*y*以弧度为单位

cos(x) *x*的余弦，*x*以弧度为单位

exp(x) *x*的指数函数

int(x) *x*的整数部分，取靠近零一侧的值

log(x) *x*的自然对数

rand( ) 比0大比1小的随机浮点值

sin(x) *x*的正弦，*x*以弧度为单位

sqrt(x) *x*的平方根

srand(x) 为计算随机数指定一个种子值

（2）除了标准数学函数外，gawk还支持一些按位操作数据的函数。

 and(v1, v2)：执行值v1和v2的按位与运算。

 compl(val)：执行val的补运算。

 lshift(val, count)：将值val左移count位。

 or(v1, v2)：执行值v1和v2的按位或运算。

 rshift(val, count)：将值val右移count位。

 xor(v1, v2)：执行值v1和v2的按位异或运算。

（3）字符穿函数

asort(s [,d]) 将数组s按数据元素值排序。索引值会被替换成表示新的排序顺序的连续数字。另外， 如果指定了d，则排序后的数组会存储在数组d中

asorti(s [,d]) 将数组s按索引值排序。生成的数组会将索引值作为数据元素值，用连续数字索引来表明排序顺序。另外如果指定了d，排序后的数组会存储在数组d中

gensub(r, s, h [, t]) 查找变量$0或目标字符串t（如果提供了的话）来匹配正则表达式r。如果h是一个以g 或G开头的字符串，就用s替换掉匹配的文本。如果h是一个数字，它表示要替换掉第h处r匹配的地方

gsub(r, s [,t]) 查找变量$0或目标字符串t（如果提供了的话）来匹配正则表达式r。如果找到了，就全部替换成字符串s

index(s, t) 返回字符串t在字符串s中的索引值，如果没找到的话返回0

length([s]) 返回字符串s的长度；如果没有指定的话，返回$0的长度

match(s, r [,a]) 返回字符串s中正则表达式r出现位置的索引。如果指定了数组a，它会存储s中匹配正则表达式的那部分

split(s, a [,r]) 将s用FS字符或正则表达式r（如果指定了的话）分开放到数组a中。返回字段的总数

sprintf(format, variables) 用提供的format和variables返回一个类似于printf输出的字符串

sub(r, s [,t]) 在变量$0或目标字符串t中查找正则表达式r的匹配。如果找到了就用字符串s替换掉第一处匹配

substr(s, i [,n]) 返回s中从索引值i开始的n个字符组成的子字符串。如果未提供n，则返回s剩下的部分

tolower(s) 将s中的所有字符转换成小写

toupper(s) 将s中的所有字符转换成大写

（4）时间函数

mktime(datespec) 将一个按YYYY MM DD HH MM SS [DST]格式指定的日期转换成时间戳值①

strftime(format [,timestamp]) 将当前时间的时间戳或timestamp（如果提供了的话）转化格式化日期（采用shell函数date()的格式）

systime( ) 返回当前时间的时间戳

（5）要定义自己的函数，必须用function关键字。

function name([variables])

{

statements

}

\_\_\_\_\_\_\_\_END\_\_\_\_\_\_\_\_